Application Program Interface Supplement
to the
Software Communications Architecture Specification

APPENDIX G

Logical Link Control (LLC)
Building Block Service Definition

Revision Summary

1.0	Initial Release

Table of Contents

Table of Figures

Table of Tables

G.1 INTRODUCTION.

G.1.1 Overview.

This document is a mapping of SCA compliant interfaces to functionality specified in the Data Link Provider Interface (DLPI) specification[1].

DLPI specifies an SCA conformant API that is an instantiation of the ISO Data Link Service Definition DIS 8886 and Logical Link Control DIS 8802-2 (LLC). Where the two standards do not conform, DIS 8886 prevails.

G.1.2 Service Layer Description.

The role of this API is to provide an interface for a complete API for a Data Link Service (DLS) provider that can support any implementation of a link layer service.

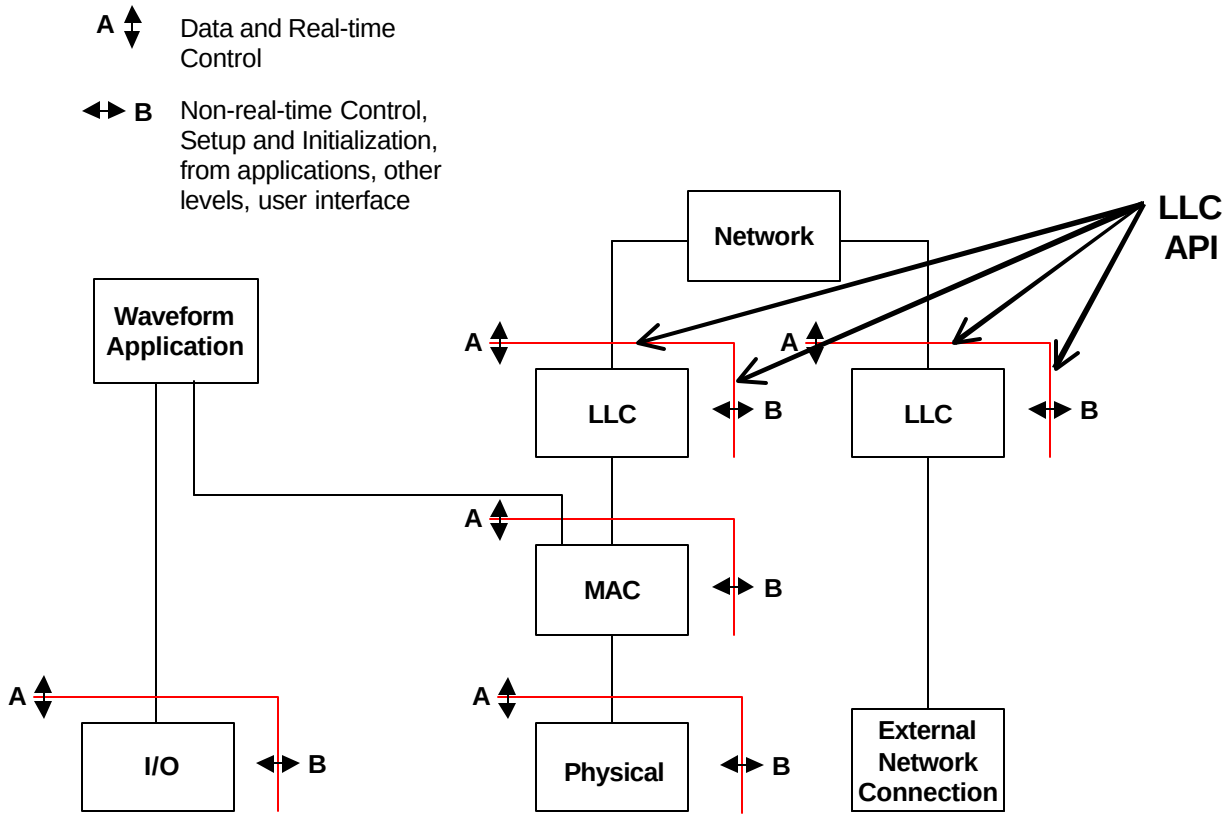

Figure 1. Service Definition Overview

G.1.3 Modes of Service.

The LLC interface supports three modes of communication: connection, connectionless and acknowledged connectionless. The connection mode is circuit-oriented and enables data to be transferred over a pre-established connection in a sequenced manner. Data may be lost or

[1] Data Link Provider Interface Specification, Revision 2.0.0, August 20, 1991, copyright © 1991 Unix International, Inc.

corrupted in this service mode, however, due to provider-initiated resynchronization or connection aborts.

The connectionless mode is message-oriented and supports data transfer in self-contained units with no logical relationship required between units. Because there is no acknowledgement of each data unit transmission, this service mode can be unreliable in the most general case. However, a specific DLS provider can provide assurance that messages will not be lost, duplicated, or reordered.

The acknowledged connectionless mode provides the means by which a data link user can send data and request the return of data at the same time. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station. The data unit transfer is point-to-point.

G.1.3.1 Connection Oriented.

The connection-mode service is characterized by four phases of communication: local management, connection establishment, data transfer, and connection release.

G.1.3.1.1 Local Management.

This phase enables a DLS user to initialize a stream for use in communication and establish an identity with the DLS provider.

G.1.3.1.2 Connection Establishment.

This phase enables two DLS users to establish a data link connection between them to exchange data. One user (the calling DLS user) initiates the connection establishment procedures, while another user (the called DLS user) waits for incoming connect requests. The called DLS user is identified by an address associated with its stream (as will be discussed shortly).

A called DLS user may either accept or deny a request for a data link connection. If the request is accepted, a connection is established between the DLS users and they enter the data transfer phase.

For both the calling and called DLS users, only one connection may be established per stream. Thus, the stream is the communication endpoint for a data link connection.

The called DLS user may choose to accept a connection on the stream where it received the connect request, or it may open a new stream to the DLS provider and accept the connection on this new, responding stream. By accepting the connection on a separate stream, the initial stream can be designated as a listening stream through which all connect requests will be processed. As each request arrives, a new stream (communication endpoint) can be opened to handle the connection, enabling subsequent requests to be queued on a single stream until they can be processed.

G.1.3.1.3 Data Transfer.

In this phase, the DLS users are considered peers and may exchange data simultaneously in both directions over an established data link connection. Either DLS user may send data to its peer DLS user at any time. Data sent by a DLS user is guaranteed to be delivered to the remote user in the order in which it was sent.

G.1.3.1.4 Connection Release.

This phase enables either the DLS user, or the DLS provider, to break an established connection. The release procedure is considered abortive, so any data that has not reached the destination user when the connection is released may be discarded by the DLS provider.

G.1.3.2 Connectionless Mode.

The connectionless mode service does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, however, the connectionless data transfer phase is immediately entered. Because there is no established connection, however, the connectionless data transfer phase requires the DLS user to identify the destination of each data unit to be transferred. The destination DLS user is identified by the address associated with that user (as will be discussed shortly).

Connectionless data transfer does not guarantee that data units will be delivered to the destination user in the order in which they were sent. Furthermore, it does not guarantee that a given data unit will reach the destination DLS user, although a given DLS provider may provide assurance that data will not be lost.

G.1.3.3 Acknowledged Connectionless Mode.

The acknowledged connectionless mode service also does not use the connection establishment and release phases of the connection-mode service. The local management phase is still required to initialize a stream. Once initialized, the acknowledged connectionless data transfer phase is immediately entered.

Acknowledged connectionless data transfer guarantees that data units will be delivered to the destination user in the order in which they were sent. A data link user entity can send a data unit to the destination DLS user, request a previously prepared data unit from the destination DLS user, or exchange data units.

G.1.4 Service States.

The following table describes the states associated with the LLC interface. It presents the state used in the state transition table, Table 1, and throughout this specification, a brief description of the state, and an indication of whether the state is valid for connection-oriented data link service (SM_CODLS), connectionless data link service (SM_CLDLS), acknowledged connectionless data link service (SM_ACLDLS) or all.

Table 1. LLC States

LLC STATE	DESCRIPTION	MODE
STATE_UNATTACHED	Component instantiated but (physical point of attachment) PPA not attached	ALL
STATE_UNBOUND	Stream is attached but not bound to a DLSAP	ALL
STATE_IDLE	The stream is bound and activated for use - connection establishment or connectionless data transfer may take place	ALL
STATE_OUTCON_PENDING	An outgoing connection is pending -the DLS user is waiting for a DL_CONNECT_CON	SM_CODLS
STATE_INCON_PENDING	An incoming connection is pending -the DLS provider is waiting for a DL_CONNECT_RES	SM_CODLS
STATE_CONN_RES_PENDING	The DLS user is waiting for an acknowledgement of a DL_CONNECT_RES	SM_CODLS
STATE_DATAXFER	Connection-mode data transfer may take place	SM_CODLS
STATE_USER_RESET_PENDING	A user-initiated reset is pending - the DLS user is waiting for a DL_RESET_CON	SM_CODLS
STATE_PROV_RESET_PENDING	A provider-initiated reset is pending -the DLS provider is waiting for a DL_RESET_RES	SM_CODLS
STATE_RESET_RES_PENDING	The DLS user is waiting for an acknowledgement of a DL_RESET_RES	SM_CODLS
STATE_DISCON_PENDING_OUTCON	The DLS user is waiting for an acknowledgement of a DL_DISCONNECT_REQ issued from the DL_OUTCON_PENDING state	SM_CODLS
STATE_DISCON_PENDING_INCON	The DLS user is waiting for an acknowledgement of a DL_DISCONNECT_REQ issued from the DL_INCON_PENDING state	SM_CODLS
STATE_DISCON_PENDING_DATAXFER	The DLS user is waiting for an acknowledgement of a DL_DISCONNECT_REQ issued from the DL_DATAXFER state	SM_CODLS
STATE_DISCON_PENDING_USER_RESET	The DLS user is waiting for an acknowledgement of a DL_DISCONNECT_REQ issued from the DL_USER_RESET_PENDING state	SM_CODLS
STATE_DISCON_PENDING_PROV_RESET	The DLS user is waiting for an acknowledgement of a DL_DISCONNECT_REQ issued from the DL_PROV_RESET_PENDING state	SM_CODLS

G.1.5 Referenced Documents.

ISO/IEC 8886 1996	Information Technology - Open Systems Interconnection - Data Link Service Definition
ISO/IEC 8802-2 1998	LANS - Part 2: Logical Link Control

G.2 UUID.

Not applicable.

G.3 SERVICES.

The features of the Connectionless LLC interface are defined in terms of the services provided by the DLS provider, and the individual primitives that may flow between the Service User and DLS provider.

The services are tabulated below and described more fully in the remainder of this section.

Table 2. Cross-Reference of Services and Primitives

Service Group	Service	Primitives
Local Management	Information Reporting	DL_INFO_REQ, DL_INFO_ACK, DL_ERROR_ACK
	Bind	DL_BIND_REQ, DL_SUBS_BIND_REQ, DL_UNBIND_REQ, DL_SUBS_UNBIND_REQ,
	Other	DL_ENABMULTI_REQ, DL_DISABMULTI_REQ, DL_PROMISCON_REQ, DL_PROMISCOFF_REQ
Connectionless Mode Data Transfer	Data Transfer	DL_UNITDATA_REQ, DL_UNITDATA_IND
	Error Reporting	DL_UDERROR_IND
Acknowledged Connectionless Mode Data Transfer	Data Transfer	DL_DATA_ACK_REQ, DL_DATA_ACK_IND, DL_DATA_ACK_STATUS_IND, DL_REPLY_REQ, DL_REPLY_IND, DL_REPLY_UPDATE_REQ, DL_REPLY_UPDATE_STATUS

G.3.1 Local Management services.

The local management services apply to all modes of service. These services, which fall outside the scope of standards specifications, define the method for initializing a stream that is connected to a DLS provider. DLS provider information reporting services are also supported by the local management facilities.

G.3.1.1 Information Reporting Service.

This service provides information about the instantiation of the link layer to the Service User. The primitive DL_INFO_REQ requests the DLS provider to return operating information about the instantiated link layer. The DLS provider returns the response in the same primitive.

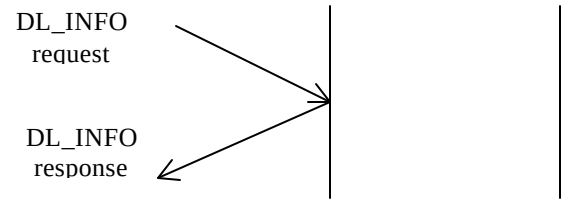

Figure 2. Message Flow - Information Reporting

G.3.1.2 Bind Service.

The bind service associates a data link service access point (DLSAP) with a stream. The DLSAP is identified by a DLSAP address.

DL_BIND_REQ requests that the DLS provider bind a DLSAP to a stream. It also notifies the DLS provider to make the stream active with respect to the DLSAP for processing connectionless and acknowledged connectionless data transfer and connection establishment requests. Protocol-specific actions taken during activation should be described in DLS provider-specific addenda.

The DLS provider returns the binding response in the same primitive. The DLS provider indicates failure by raising and exception.

Certain DLS providers require the capability of binding on multiple DLSAP addresses. DL_SUBS_BIND_REQ provides that added capability. The DLS provider returns the bound DLSAP address in the same primitive. The DLS provider indicates failure by raising and exception.

The normal flow of messages is illustrated in Figure 3.

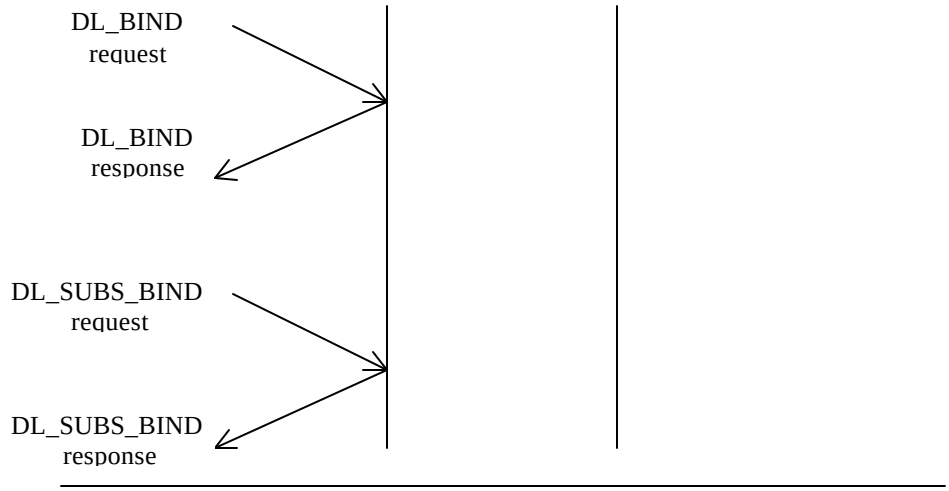

Figure 3. Message Flow - Binding a Stream to a DLSAP

DL_UNBIND_REQ requests the DLS provider to unbind all DLSAP(s) from a stream. The DL_UNBIND_REQ also unbinds all the subsequently bound DLSAPs that have not been unbound. The DLS provider indicates failure by raising an exception.

DL_SUBS_UNBIND_REQ requests the DLS Provider to unbind the subsequently bound DLSAP. The DLS Provider indicates failure by raising an exception.

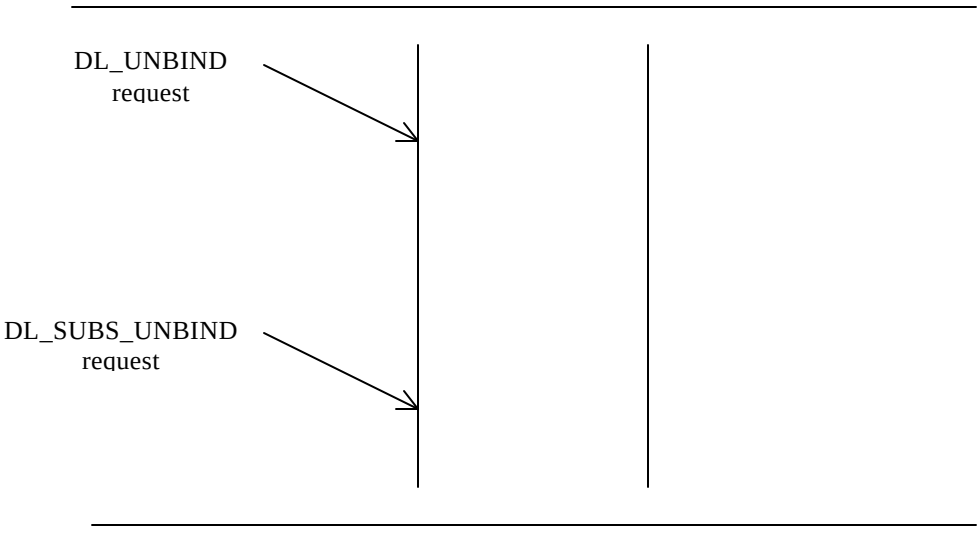

Figure 4. Message Flow - Unbinding a Stream from a DLSAP

DL_ENABMULTI_REQ requests the DLS Provider to enable specific multicast addresses on a per stream basis. The DLS provider indicates failure by raising an exception.

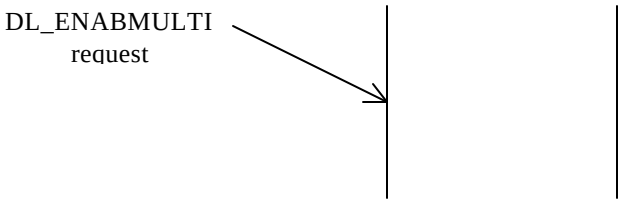

Figure 5. Message Flow: Enabling a specific multicast address on a Stream

DL_DISABMULTI_REQ requests the DLS Provider to disable specific multicast addresses on a per stream basis. The DLS provider indicates failure by raising an exception.

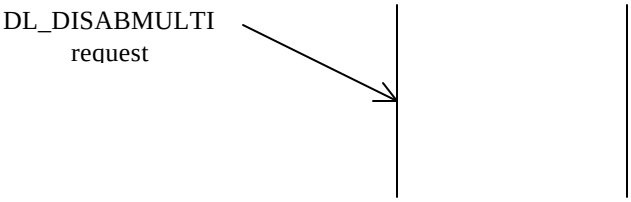

Figure 6. Message Flow: Disabling a specific multicast address on a Stream

DL_PROMISCON_REQ requests the DLS Provider to enable promiscuous mode on a per Stream basis, either at the physical level or at the Service Access Point (SAP) level. The DLS provider indicates failure by raising an exception.

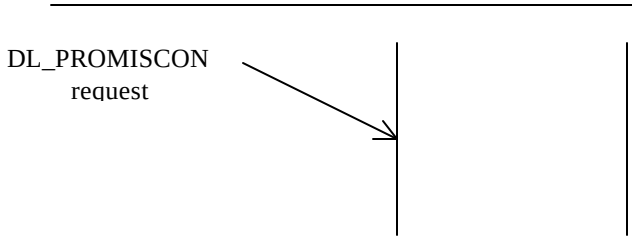

Figure 7. Message Flow: Enabling promiscuous mode on a Stream

DL_PROMISCOFF_REQ requests the DLS Provider to disable promiscuous mode on a per Stream basis, either at the physical level or at the SAP level. The DLS provider indicates failure by raising an exception.

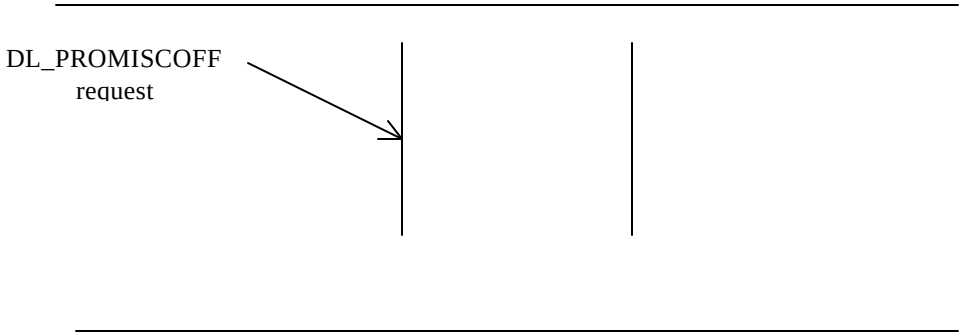

Figure 8. Message Flow: Disabling promiscuous mode on a Stream

G.4 SERVICE PRIMITIVES.

The interface to the local management portion of the data link layer defines an IDL interface between the provider of the data link service (DLS provider) and the consumer of the data link service (DLS user). This interface in combination with other link layer interfaces will form a complete link layer API. Figure 9 shows the LLC dependencies on other SCA elements.

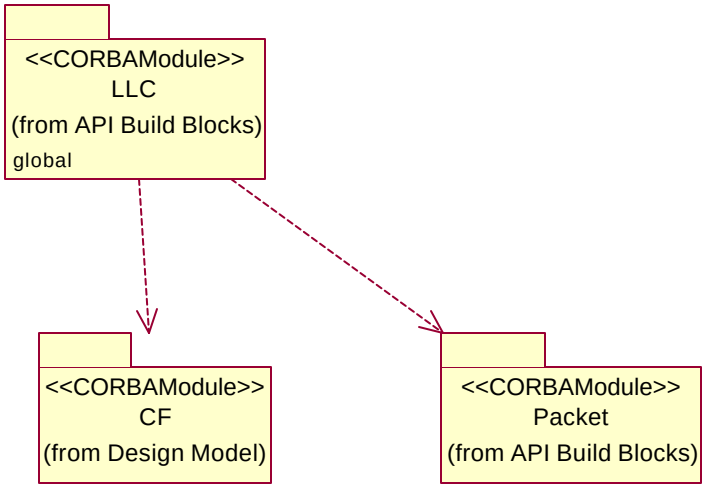

Figure 9. LLC Dependencies

Figure 10 shows the types that are common across the LLC.

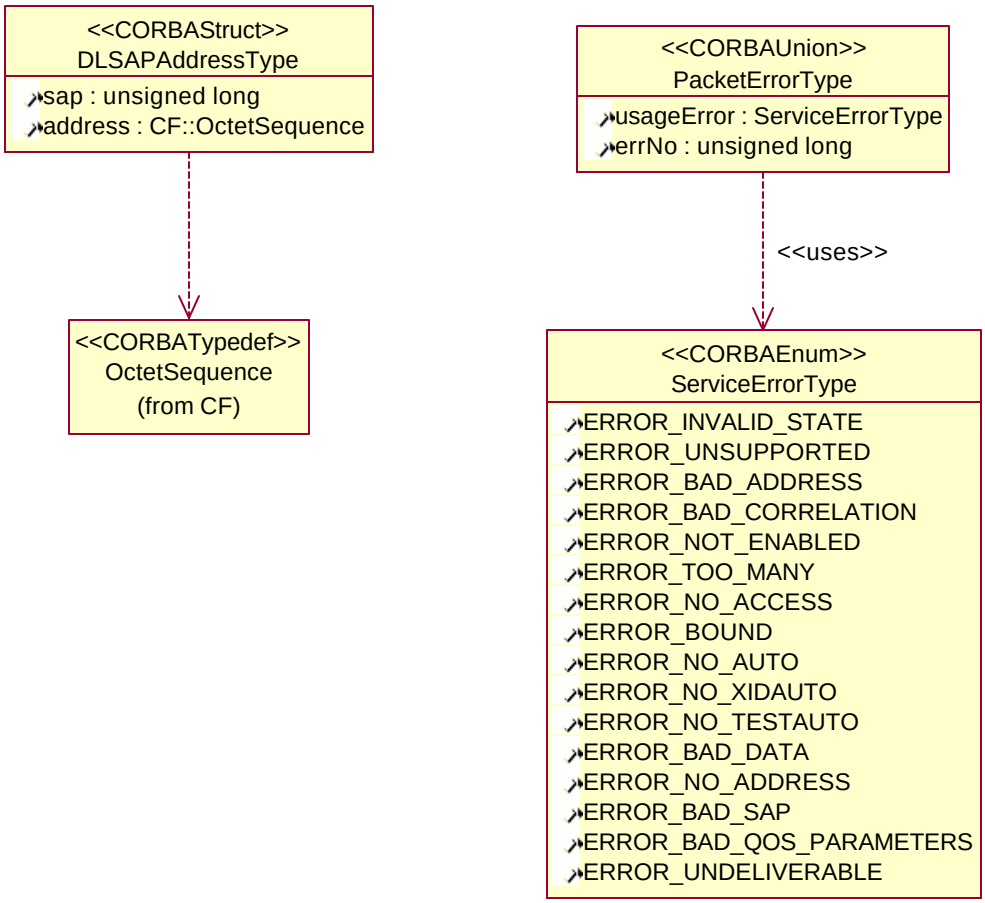

Figure 10. Class Diagram: LLC Common Types

G.4.1 Local management service primitives.

This section describes the local management service primitives that are common to connection, connectionless and acknowledged connectionless service modes in full link layer API. These primitives support the Information Reporting, Bind, enabling/disabling of multicast addresses and turning on/off the promiscuous mode. Once a connection has been established between the DLS provider and the Service User, these primitives initialize the layer, preparing it for use.

The class diagrams in Figure 11, Figure 12 and Figure 13 show the enumeration types, structures and interfaces defined for the Local Management service.

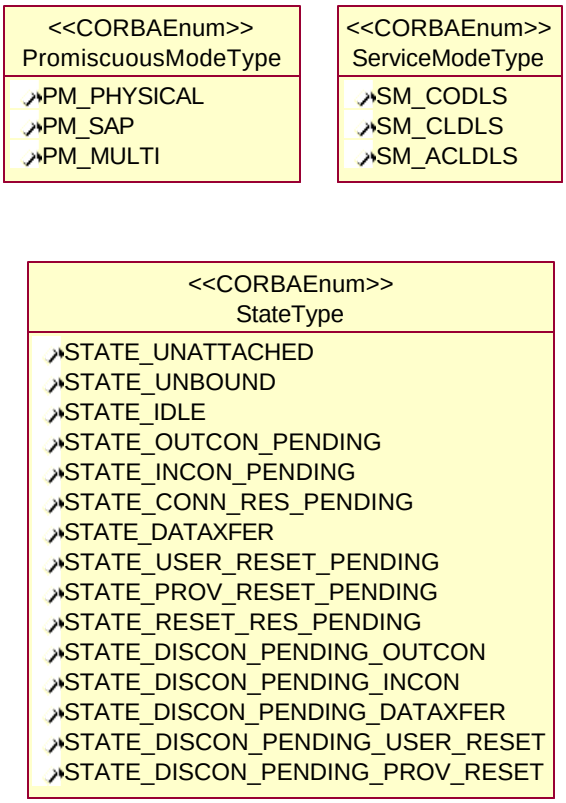

Figure 11. Local Management Class Diagram: Enumerations

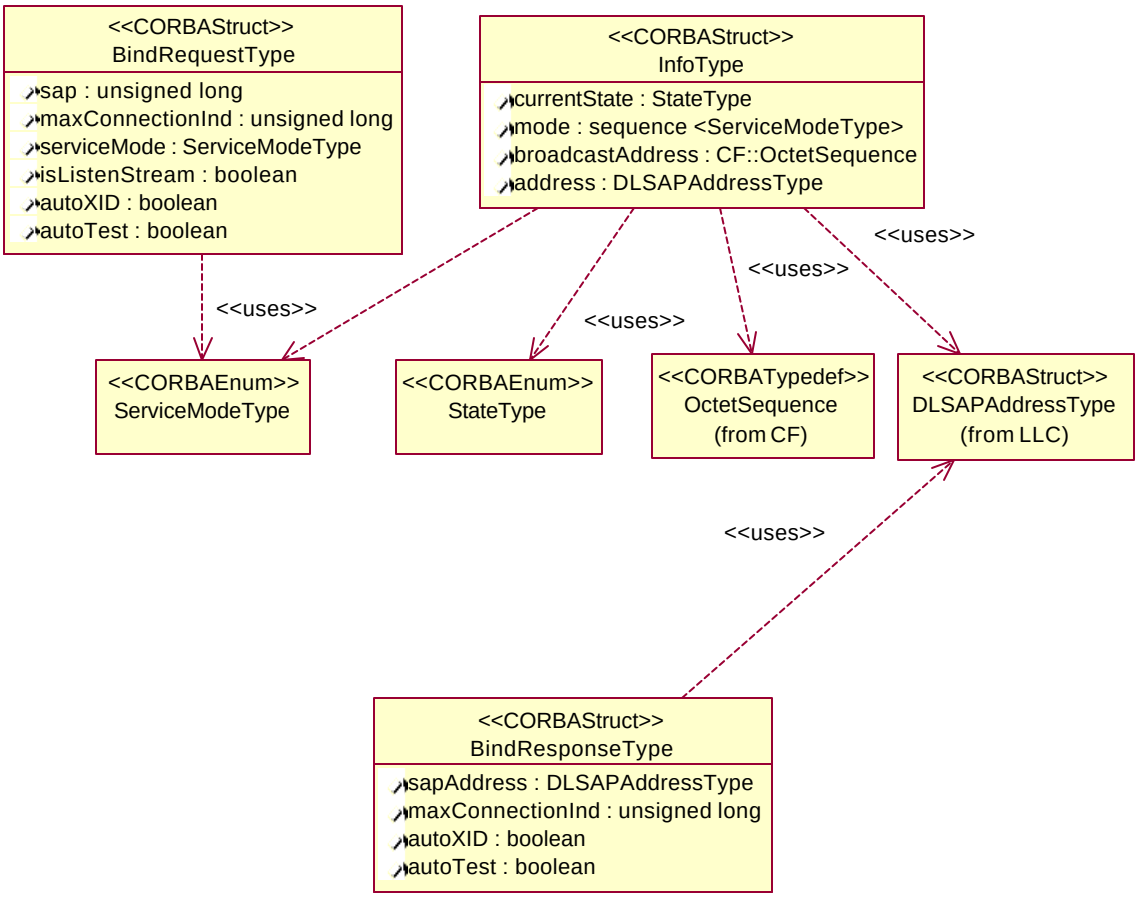

Figure 12. Local Management Class Diagram: Structures

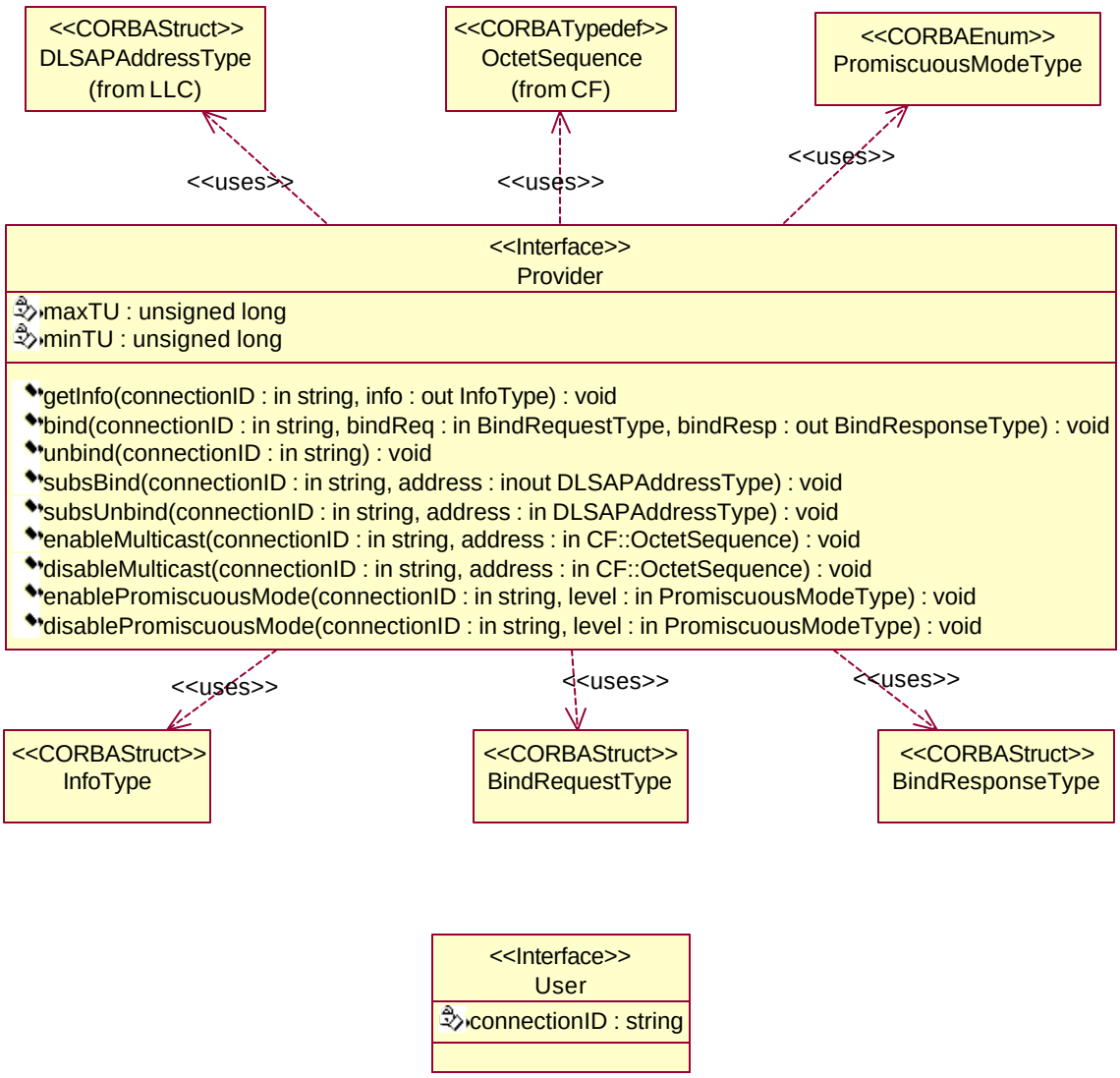

Figure 13. Local Management Class Diagram: Provider and User Interfaces

G.4.1.1 PPA Initialization/De-initialization.

The Physical Point of Attachment (PPA) associated with each stream must be initialized before the DLS provider can transfer data over the medium. The initialization and de-initialization of the PPA is a network management issue, but this service must address the issue because of the impact such actions will have on a DLS user. More specifically, this service requires the DLS provider to initialize the PPA associated with a stream at some point before it completes the processing of the DL_BIND_REQ. Guidelines for initialization and de-initialization of a PPA by a DLS provider are presented here.

A DLS provider may initialize a PPA using the following methods:

- pre-initialized by some network management mechanism before the DL_BIND_REQ is received; or
- automatic initialization on receipt of a DL_BIND_REQ or the configure command through the resource interface.

A specific DLS provider may support either of these methods, or possibly some combination of the two, but the method implemented has no impact on the DLS user. From the DLS user's viewpoint, the PPA is guaranteed to be initialized after successful execution of the DL_BIND_REQ. For automatic initialization, this implies that the DL_BIND_REQ cannot complete until the initialization has completed.

If pre-initialization has not been performed and/or automatic initialization fails, the DLS provider will fail the DL_BIND_REQ. Two errors, ERROR_INITFAILED and ERROR_NOTINIT, may be returned in the ServiceUsageError exception raised by a DL_BIND_REQ if PPA initialization fails. ERROR_INITFAILED is returned when a DLS provider supports automatic PPA initialization, but the initialization attempt failed. ERROR_NOTINIT is returned when the DLS provider requires pre-initialization, but the PPA is not initialized before the DL_BIND_REQ is received.

A DLS provider may handle PPA de-initialization using the following methods:

- automatic de-initialization upon receipt of the DL_UNBIND_REQ or upon disconnection of the last stream associated with the PPA;
- automatic de-initialization after expiration of a timer following the last DL_UNBIND_REQ, or close as appropriate; or
- no automatic de-initialization; administrative intervention is required to de-initialize the PPA at some point after it is no longer being accessed.

A specific DLS provider may support any of these methods, or possibly some combination of them, but the method implemented has no impact on the DLS user. From the DLS user's viewpoint, the PPA is guaranteed to be initialized and available for transmission until it closes or unbinds the stream associated with the PPA.

DLS provider-specific addendum documentation should describe the method chosen for PPA initialization and de-initialization.

G.4.1.2 Stream Connection.

The connection of a set of components to form a bi-directional data path is referred to in this document as a stream. A given component may de-multiplex between several other components. Each of the connections to the other components represents an individual stream.

The DLS provider must be able to de-multiplex between multiple streams (DLS users). For this to occur the DLS Provider must be able to identify the source of operation invocations as well as associate a Service Access Point (SAP) with a stream. The diagram in Figure 14 illustrates the sequence of events that must occur for the DLS provider to be properly connected to a DLS user.

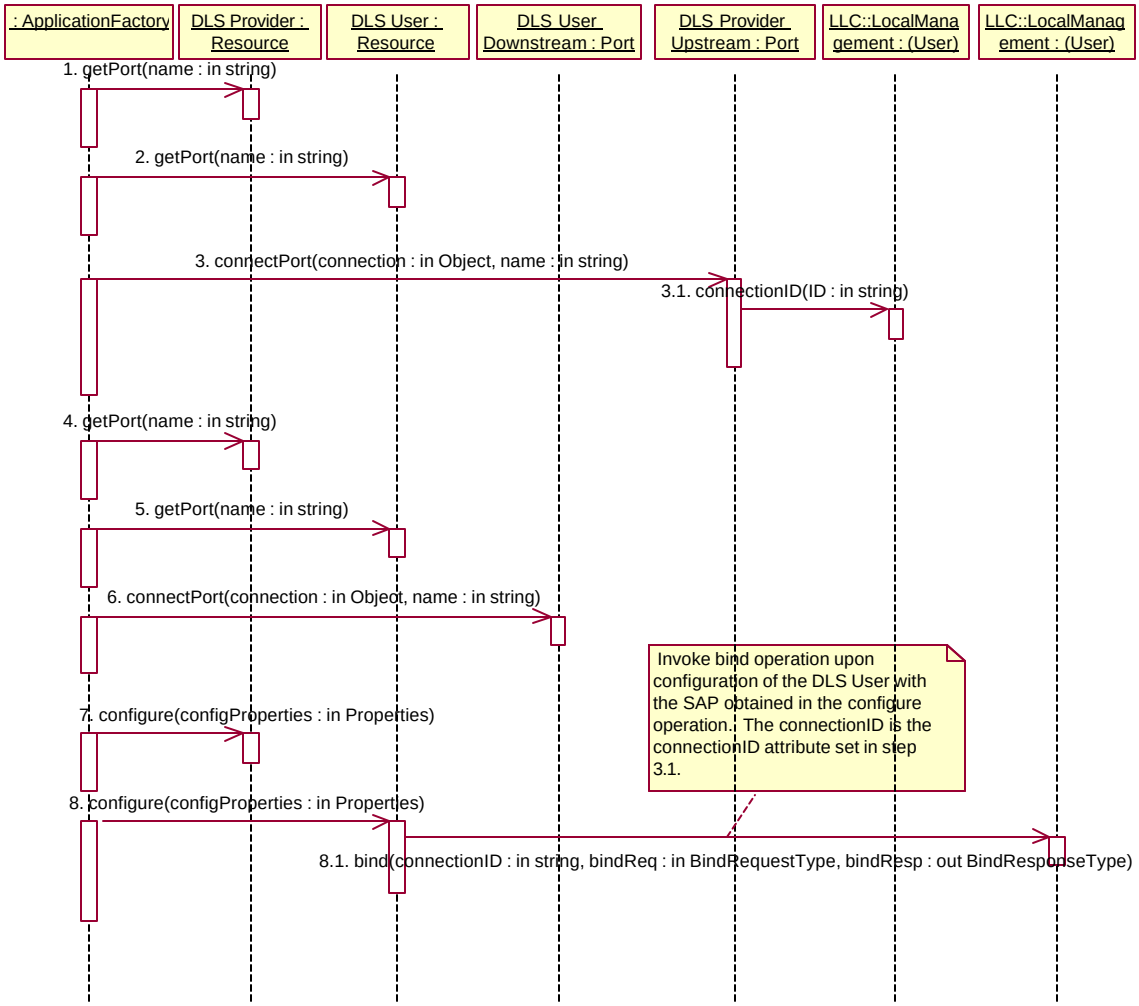

Figure 14. Stream Connection: Sequence of events

1 The CF::*ApplicationFactory* invokes *getPort* on the DLS Provider resource to get the object that uses the LLC::LocalManagement::User interface.

2 The *ApplicationFactory* invokes *getPort* on the DLS User resource to get the object that realizes the LLC::LocalManagement::User interface.

3 The *ApplicationFactory* invokes *connectPort* on the object obtained in step 2 from the DLS Provider with the object in step 1 from the DLS User as a parameter.

3.1 The implementation of *connectPort* for the object obtained from the DLS User in step 1 sets the connectionID attribute on the object obtained in step 2 that realizes the LLC::LocalManagement::User interface. This step identifies to the DLS User its connectionID associated with the DLS Provider for upstream data flow.

4 The *ApplicationFactory* invokes *getPort* on the DLS Provider resource to get the object reference that realizes the LLC::LocalManagement::Provider interface.

5 The *ApplicationFactory* invokes *getPort* on the DLS User resource to get the object that uses the LLC::LocalManagement::Provider interface.

6 The *ApplicationFactory* invokes *connectPort* on the DLS User to connect the in LLC::LocalManagement::Provider interface to the DLS User.

7 The *ApplicationFactory* invokes *configure* on the DLS Provider resource.

8 The *ApplicationFactory* invokes *configure* on the DLS User resource with the SAP(s) to be configured

8.1 The DLS User resource invokes the bind operation on the object obtained in step 4 that realizes the LLC::LocalManagement::Provider interface. This operation associates the SAP of the DLS User (SAPConfiguratorType) with its object reference in the DLS provider.

Upon completion of this sequence of events the DLS provider can now de-multiplex upstream data between multiple DLS Users based on traffic content based on the SAP. If there are multiple SAPs associated with the stream the DLS User will invoke the subsBind operation.

G.4.1.3 DL_MAX_SDU_REQ.

This primitive requests the maximum size of a single service data unit (SDU) of the physical/logical interface to which the DLS provider is attached.

G.4.1.3.1 Synopsis.

```
readonly attribute unsigned long maxSDU
```

This attribute will turn into an accessor function in the target language.

G.4.1.3.2 Parameters.

.N/A

G.4.1.3.3 State.

This primitive is valid in any state.

G.4.1.3.4 New State.

No change.

G.4.1.3.5 Response.

The DLS provider returns an unsigned long value indicating the maximum size of a single SDU of the physical interface/logical interface to which the link layer is attached.

G.4.1.3.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.3.7 Errors/Exceptions.

N/A

G.4.1.4 DL_MIN_SDU_REQ.

This primitive requests the minimum size of a single SDU of the physical/logical interface to which the DLS provider is attached.

G.4.1.4.1 Synopsis.

```
readonly attribute unsigned long minSDU
```

This attribute will turn into an accessor function in the target language.

G.4.1.4.2 Parameters.

N/A.

G.4.1.4.3 State.

This primitive is valid in any state.

G.4.1.4.4 New State.

No change.

G.4.1.4.5 Response.

The DLS provider returns an unsigned long value indicating the minimum size of a single SDU of the physical interface/logical interface to which the link layer is attached.

G.4.1.4.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.4.7 Errors/Exceptions.

N/A.

G.4.1.5 DL_INFO_REQ.

This primitive requests information of the DLS provider about the stream. This information includes a set of provider-specific parameters, as well as the current state of the interface.

G.4.1.5.1 Synopsis.

void getInfo (in string connectionID, out InfoType info) raises (InvalidPort, SystemError);

G.4.1.5.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

G.4.1.5.3 State.

The message is valid in any state.

G.4.1.5.4 New State.

The resulting state is unchanged.

G.4.1.5.5 Response.

The DLS provider returns the information about the stream as an out parameter in the info parameter which has the following structure:

```
struct InfoType {
        StateType                       currentState;
        sequence <ServiceModeType       mode;
        CF::OctetSequence               broadcastAddress;
        DLSAPAddressType                address;
};
```

currentState

conveys the state of the interface for the connection when the DLS provider issued this acknowledgement. See Appendix B for a list of the states and an explanation of each.

mode

if returned before the DL_BIND_REQ is processed, this conveys which service modes (connection-mode, connectionless-mode or acknowledged connectionless-mode, or any combination of these modes) the DLS provider can support. It contains a sequence specifying one or more than one of the following values:

CODLS connection-oriented data link service;
CLDLS connectionless data link service;
ACLDLS acknowledged connectionless data link service;

Once a specific service mode has been bound to the connection, this field returns that specific service mode.

broadcastAddress

this is the physical broadcast address

address

conveys the address that is bound to the associated connectionID. If the DLS user issues a DL_INFO_REQ prior to binding a DLSAP, the content of address is undefined.

G.4.1.5.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.5.7 Errors/Exceptions.

If the request fails, an exception will be generated.

InvalidPort

The connectionID was not recognized by the DLS provider.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.6 DL_BIND_REQ.

Requests the DLS provider bind a DLSAP to the stream. The DLS user must identify the address of the DLSAP to be bound to the stream. For connection-mode service, the DLS user also indicates whether it will accept incoming connection requests on the stream. Finally, the request directs the DLS provider to activate the stream associated with the DLSAP. A stream is viewed as active when the DLS provider may transmit and receive protocol data units destined to or originating from the stream. The PPA associated with each stream must be initialized upon completion of the processing of the DL_BIND_REQ (see section 4.1.1, *PPA Initialization / De-initializatio*n). More specifically, the DLS user is ensured that the PPA is initialized when the DL_BIND_REQ is completed successfully. If the PPA cannot be initialized, the DL_BIND_REQ will fail. A stream may be bound as a "connection management" stream, such that it will receive all connect requests that arrive through a given PPA (the connection management stream will be explained in more detail in a future revision of this document when the connection oriented service will be included). In this case, sap will be ignored.

G.4.1.6.1 Synopsis.

```
void bind (
        in string                   connectionID,
        in BindRequestType          bindReq,
        out bindResponseType        bindResp
) raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.6.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

bindReq

the bindReq has the following structure:

```
struct BindRequestType {
        any                 sap;
        unsigned long       maxConnectionInd;
        ServiceModeType     serviceMode;
        boolean             isListenStream;
        boolean             autoXID;
        boolean             autoTest;
};
```

sap

conveys sufficient information to identify the DLSAP that will be bound to the DLPI stream (see section 2.3, *DLPI Addressin*g, for a description of DLSAP addresses). The format of this information is specific to a given DLS provider, and may contain the full DLSAP address or some portion of that address

sufficient to uniquely identify the DLSAP in question. The full address of the bound DLSAP will be returned in the bindResp.

The following rules are used by the DLS provider when binding a DLSAP address.

- The DLS provider must define and manage its DLSAP address space.

- This service allows the same DLSAP to be bound to multiple connectionIDs, but a given DLS provider may need to restrict its address space to allow one stream per DLSAP.

- The DLS provider may not be able to bind the specified DLSAP address for the following reasons:

  (1) the DLS provider may statically associate a specific DLSAP with each stream; or

  (2) the DLS provider may only support one stream per DLSAP and the DLS user attempted to bind a DLSAP that was already bound to another stream.

In case (1), the value of *sap* is ignored by the DLS provider and the bindResp returns the DLSAP address that is already associated with the stream. In case (2), if the DLS provider cannot bind the given DLSAP to the stream, it may attempt to choose an alternate DLSAP and return that on the DL_BIND_ACK. If an alternate DLSAP cannot be chosen, the DLS provider will raise ServiceUsageError with *qualifier* set to NOADDR.

Because of the provider-specific nature of the DLSAP address, DLS user software that is to be protocol independent should avoid hard-coding this value. The DLS user should retrieve the necessary DLSAP address from some other entity (such as a management entity or higher layer protocol entity) and insert it without inspection into the bindReq.

maxConnectionInd

conveys the maximum number of outstanding DL_CONNECT_IND messages allowed on the stream. If the value is zero, the stream cannot accept any DL_CONNECT_IND messages. If greater than zero, the DLS user will accept DL_CONNECT_IND messages up to the given value before having to respond with a DL_CONNECT_RES or a DL_DISCONNECT_REQ (a future section on *Multi-threaded Connection Establishmen*t, will provide details on how this value is used to support multi-threaded connect processing). The DLS provider may not be able to support the value supplied in maxConnectionInd, as specified by the following rules.

- If the provider cannot support the specified number of outstanding connect indications, it should set the value down to a number it can support.

- Only one stream that is bound to the indicated DLSAP may have an allowed number of maximum outstanding connect indications greater than zero. If a DL_BIND_REQ specifies a value greater than zero, but another stream has

already bound itself to the DLSAP with a value greater than zero, the DLS provider will fail the request, raising *ServiceUsageError* with *qualifier* set to BOUND.

- If a stream with maxConnectionInd greater than zero is used to accept a connection, the stream will be found busy during the duration of the connection, and no other streams may be bound to the same DLSAP with a value of maxConnectionInd greater than zero. This restriction prevents more than one stream bound to the same DLSAP from receiving connect indications and accepting connections. Accepting a connection on such a stream is only allowed if there is just a single outstanding connect indication being processed.

- A DLS user should always be able to request a maxConnectionInd value of zero, since this indicates to the DLS provider that the stream will only be used to originate connect requests.

- A stream with a negotiated value of maxConnectionInd that is greater than zero may not originate connect requests.

This field is ignored in connectionless-mode service.

serviceMode

conveys the desired mode of service for this stream, and may contain one of the following:

DL_CODLS connection-oriented data link service.

DL_CLDLS connectionless data link service.

DL_ACLDLS acknowledged connectionless data link service.

If the DLS provider does not support the requested service mode, a ServiceUsageError exception will be raised, with *qualifier* set to UNSUPPORTED.

isListenStream

if true, indicates that the stream is the "connection management" stream for the PPA to which the stream is attached. When an incoming connect request arrives, the DLS provider will first look for a stream bound with *maxConnectionInd g*reater than zero that is associated with the destination DLSAP. If such a stream is found, the connect indication will be issued on that stream. Otherwise, the DLS provider will issue the connect indication on the "connection management" stream for that PPA, if one exists. Only one "connection management" stream is allowed per PPA, so an attempt to bind a second "connection management" stream on a PPA will fail and the ServiceUsageError exception will be raised with *qualifier* error set to BOUND. When *maxConnectionInd* is non-zero, the value of *sap* will be ignored. In connectionless-mode service, *maxConnectionInd* is ignored by the DLS provider.

autoXID, autoTest

indicates to the DLS Provider that XID and/or TEST responses for this stream are to be automatically generated by the DLS Provider. The DLS Provider will not generate DL_XID_IND and/or DL_TEST_IND, and will indicate as erroneous a DL_XID_REQ and/or DL_TEST_REQ. If the DLS Provider does not support automatic handling of XID and/or TEST responses, a ServiceUsageError exception will be raised with *qualifier* set to ERROR_NO_AUTO, ERROR_NO_XIDAUTO or ERROR__NO_TESTAUTO.

G.4.1.6.3 State.

This primitive is valid in state STATE_UNBOUND.

G.4.1.6.4 New State.

The resulting state is STATE_IDLE.

G.4.1.6.5 Response.

A successful bind of a DLSAP to a stream will return information in the bindResp parameter which consists of the following structure.

```
struct BindResponseType {
        DLSAPAddressType  sapAddress;
        unsigned long     maxConnectionInd;
        unsigned long     xidTestFlag;
};
```

sapAddress

conveys the DLSAP address information associated with the bound DLSAP. It corresponds to the *sap* field of the associated DL_BIND_REQ, which contains either part or all of the DLSAP address. For that portion of the DLSAP address conveyed in the DL_BIND_REQ, this field contains the corresponding portion of the address for the DLSAP that was actually bound.

MaxConnectionInd

conveys the allowed, maximum number of outstanding DL_CONNECT_IND messages to be supported on the stream. If the value is zero, the stream cannot accept any DL_CONNECT_IND messages. If greater than zero, the DLS user will accept DL_CONNECT_IND messages up to the given value before having to respond with a DL_CONNECT_RES or a DL_DISCONNECT_REQ. The rules for negotiating this value are presented under the description of DL_BIND_REQ.

autoXID, autoTest

conveys the XID and TEST responses supported by the provider. If true the relevant response is automatically generated by the DLS provider.

G.4.1.6.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.6.7 Errors/Exceptions.

If the request fails, an exception will be generated.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_BADADDR	The DLSAP address information was invalid or was in an incorrect format.
ERROR_INITFAILED	Automatic initialization of the PPA failed.
ERROR_NOTINIT	The PPA had not been initialized prior to this request.
ERROR_ACCESS	The DLS user did not have proper permission to use the requested DLSAP address.
ERROR_BOUND	The DLS user attempted to bind a second stream to a DLSAP with maxConnectionInd greater than zero, or the DLS user attempted to bind a second "connection management" stream to a PPA.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_NOADDR	The DLS provider could not allocate a DLSAP address for this stream.
ERROR_UNSUPPORTED	The DLS provider does not support requested service mode on this stream.
ERROR_NO_AUTO	Automatic handling of XID and TEST responses not supported.
ERROR_NO_XIDAUTO	Automatic handling of XID response not supported.
ERROR_NO_TESTAUTO	Automatic handling of TEST response not supported.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.7 DL_UNBIND_REQ.

Requests the DLS provider to unbind the DLSAP that had been bound by a previous DL_BIND_REQ from this stream. If one or more DLSAPs were bound to the stream using a DL_SUBS_BIND_REQ, and have not been unbound using a DL_SUBS_UNBIND_REQ, the DL_UNBIND_REQ will unbind all the subsequent DLSAPs for that stream along with the DLSAP bound using the previous DL_BIND_REQ.

At the successful completion of the request, the DLS user may issue a new DL_BIND_REQ for a potentially new DLSAP.

G.4.1.7.1 Synopsis.

void unbind (in string connectionID) raises (InvalidPort, ServiceUsageError, SystemError);

G.4.1.7.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

G.4.1.7.3 State.

The operation is valid in STATE_IDLE.

G.4.1.7.4 New State.

The resulting state is STATE_UNBOUND.

G.4.1.7.5 Response.

N/A.

G.4.1.7.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.7.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_INVALID_STATE The primitive was issued from an invalid state.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.8 DL_SUBS_BIND_REQ.

Requests the DLS provider bind a subsequent DLSAP to the stream. The DLS user must identify the address of the subsequent DLSAP to be bound to the stream.

G.4.1.8.1 Synopsis.

```
void subsBind (
        in string                   connectionID,
        inout DLSAPAddressType      address
)  raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.8.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

address

conveys the DLSAP address to be bound.

G.4.1.8.3 State.

The primitive is valid in STATE_IDLE.

G.4.1.8.4 New State.

The resulting state is unchanged.

G.4.1.8.5 Response.

The bound DLSAP address is returned to the DLS user in the *address* parameter.

G.4.1.8.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.8.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_BADADDR	The DLSAP address information was invalid or was in an incorrect format.
ERROR_ACCESS	The DLS user did not have proper permission to use the requested DLSAP address.

ERROR_BOUND	The DLS user attempted to bind a second stream to a DLSAP with maxConnectionInd greater than zero, or the DLS user attempted to bind a second "connection management" stream to a PPA.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_UNSUPPORTED	The DLS provider does not support requested service mode on this stream.
ERROR_TOO_MANY	Limit exceeded on the number of DLSAPs per stream

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.9 DL_SUBS_UNBIND_REQ.

Requests the DLS Provider to unbind the DLSAP that had been bound by a previous DL_SUBS_BIND_REQ from this stream.

G.4.1.9.1 Synopsis.

```
void subsUnbind (
        in string                connectionID,
        in DLSAPAddressType      address
)  raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.9.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

address

identifies the DLSAP address to unbind from the stream.

G.4.1.9.3 State.

The primitive is valid in STATE_IDLE.

G.4.1.9.4 New State.

The resulting state is unchanged.

G.4.1.9.5 Response.

N/A.

G.4.1.9.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.9.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_BADADDR	The DLSAP address information was invalid or was in an incorrect format.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.10 DL_ENABMULTI_REQ.

Requests the DLS provider to enable specific multicast addresses on a per stream basis. It is invalid for a DLS provider to pass upstream messages that are destined for any address other than those explicitly enabled on that stream by the DLS user.

G.4.1.10.1 Synopsis.

```
void enableMulticast (
        in string               connectionID,
        in CF::OctetSequence address
) raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.10.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

address

the multicast address to enable.

G.4.1.10.3 State.

This primitive is valid in any state accept STATE_UNATTACHED.

G.4.1.10.4 New State.

The resulting state is unchanged.

G.4.1.10.5 Response.

N/A.

G.4.1.10.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.10.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_BADADDR	The address information was invalid or was in an incorrect format.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_UNSUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_TOO_MANY	Too many multicast address enable attempts. Limit exceeded.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.11 DL_DISABMULTI_REQ.

Requests the DLS Provider to disable specific multicast addresses on a per stream basis.

G.4.1.11.1 Synopsis.

```
void disableMulticast (
        in string               connectionID,
        in CF::OctetSequence address
)  raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.11.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

address

the multicast address to disable.

G.4.1.11.3 State.

This primitive is valid in any state accept STATE_UNATTACHED.

G.4.1.11.4 New State.

The resulting state is unchanged.

G.4.1.11.5 Response.

N/A.

G.4.1.11.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.11.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_BADADDR	The address information was invalid or was in an incorrect format.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_UNSUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_NOT_ENABLED	Address specified is not enabled.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.12 DL_PROMISCON_REQ.

This primitive requests the DLS Provider to enable promiscuous mode on a per stream basis, either at the physical level or at the SAP level.

The DL Provider will route all received messages on the media to the DLS user until either a DL_PROMISCOFF_REQ is received or the stream is torn down.

G.4.1.12.1 Synopsis.

```
void enablePromiscuousMode (
        in string                connectionID,
        in PromiscuousModeType   level
) raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.12.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

level

indicates promiscuous mode at the physical or SAP level.

PM_PHYS	indicates promiscuous mode at the physical level
PM_SAP	indicates promiscuous mode at the SAP level
PM_MULTI	indicates promiscuous mode for all multicast addresses

G.4.1.12.3 State.

This primitive is valid in any state.

G.4.1.12.4 New State.

The resulting state is unchanged.

G.4.1.12.5 Response.

N/A.

G.4.1.12.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.12.7 Errors/Exceptions.

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_NOT_SUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_UNSUPPORTED	The primitive is not supported by the DLS provider.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.1.13 DL_PROMISCOFF_REQ.

This primitive requests the DLS provider to disable promiscuous mode on a per stream basis, either at the physical level or at the SAP level.

G.4.1.13.1 Synopsis.

```
void disablePromiscuousMode (
        in string                connectionID,
        in PromiscuousModeType   level
) raises (InvalidPort, ServiceUsageError, SystemError);
```

G.4.1.13.2 Parameters.

connectionID

identifies the stream (a DLS User that is connected to the DLS Provider) .

level

indicates promiscuous mode at the physical or SAP level.

PM_PHYS	indicates promiscuous mode at the physical level
PM_SAP	indicates promiscuous mode at the SAP level
PM_MULTI	indicates promiscuous mode for all multicast addresses

G.4.1.13.3 State.

This primitive is valid in any state.

G.4.1.13.4 New State.

The resulting state is unchanged.

G.4.1.13.5 Response.

N/A.

G.4.1.13.6 Originator.

This primitive is initiated by the DLS user.

G.4.1.13.7 Errors/Exceptions

InvalidPort

The connectionID was not recognized by the DLS provider.

ServiceUsageError

Indicates that the DLS provider could not complete the request because of an incorrect use of the service. The qualifier returned in the exception can be one of the following.

ERROR_INVALID_STATE	The primitive was issued from an invalid state.

ERROR_NOT_SUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_MODE_NOT_EABLED	The primitive is not supported by the DLS provider.

SystemError

indicates an error occurred in the environment. The POSIX errno associated with error is placed in errNo. Not to be confused with CORBA system error exceptions.

G.4.2 connectionless mode service primitives.

Figure 15, Figure 16, Figure 17 and Figure 18 are the type, Packet Building Block instantiations and Provider and User class diagrams respectively for the connectionless data transfer service. The DLS user implements the Provider interface and the DLS user implements the User interface.

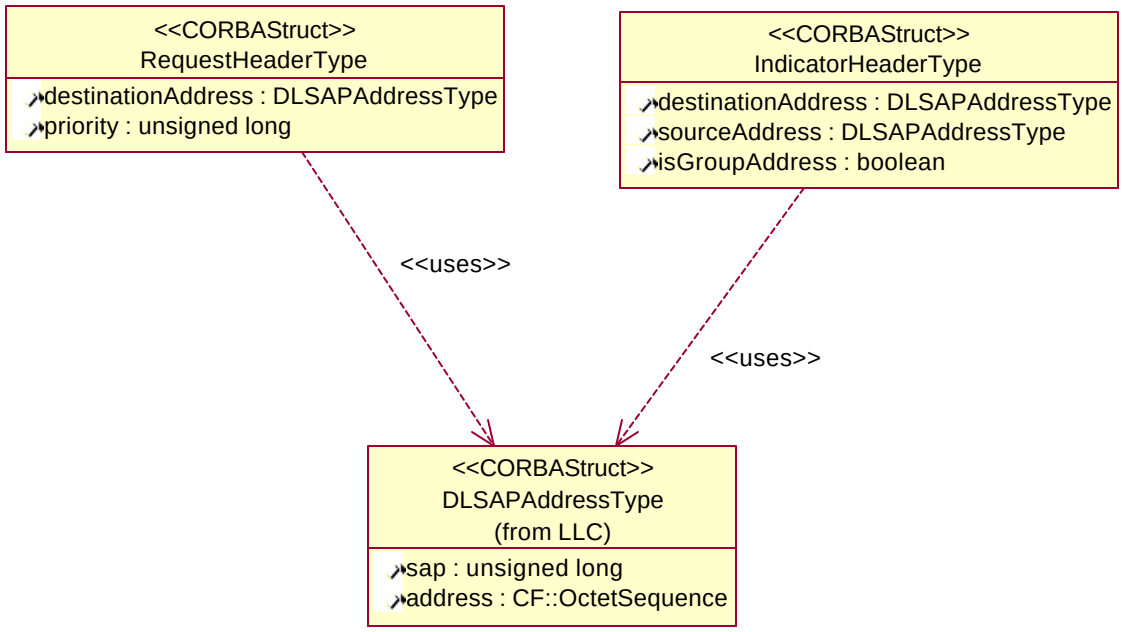

Figure 15. Connectionless Class Diagram: Types

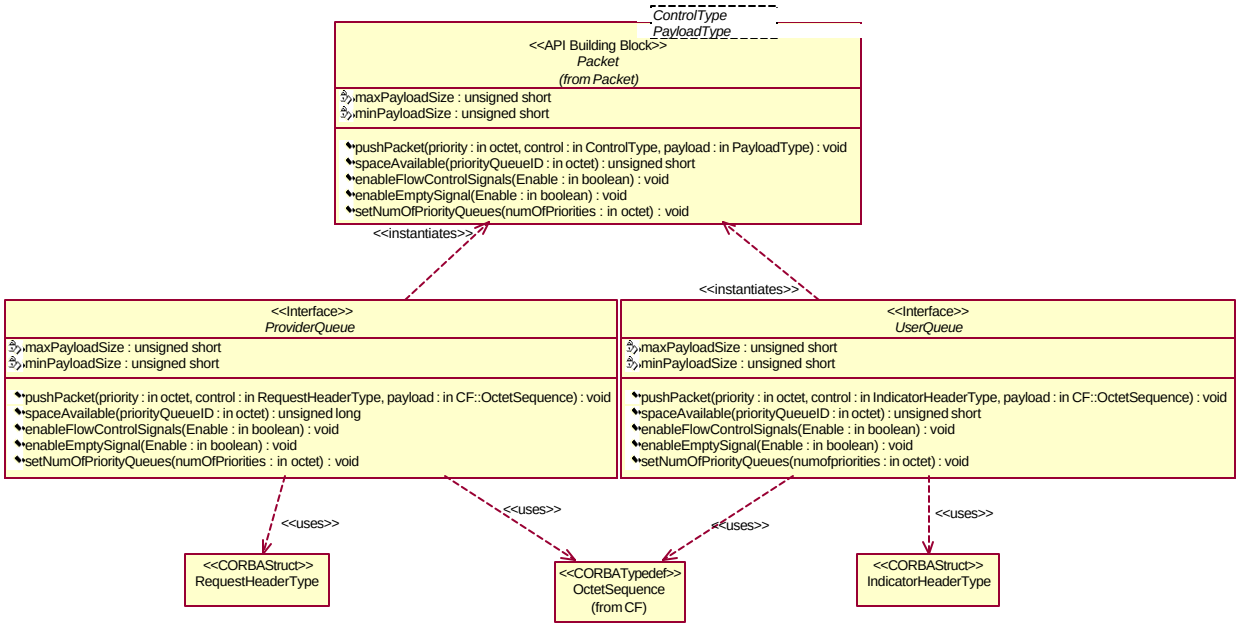

Figure 16. Connectionless Class Diagram: Instantiations of Packet Building Block

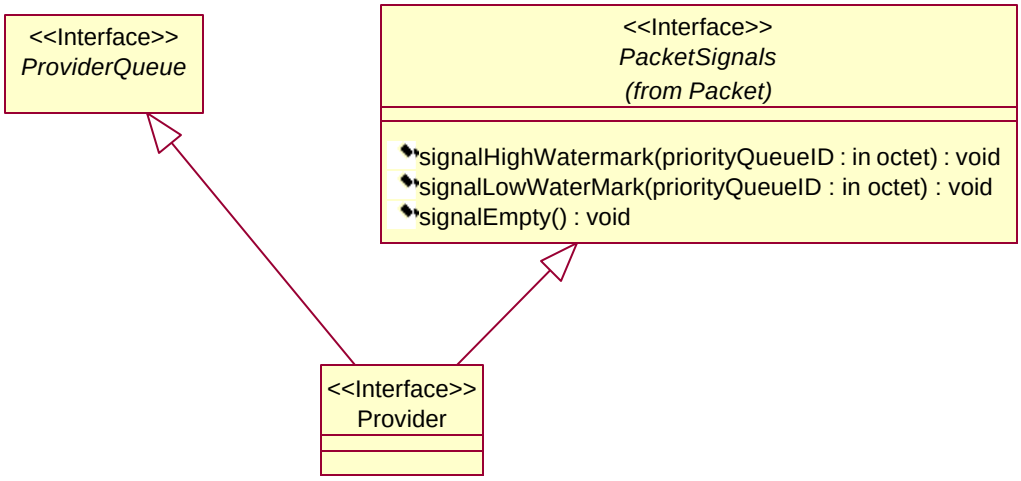

Figure 17. Connectionless Class Diagram: Provider Interface

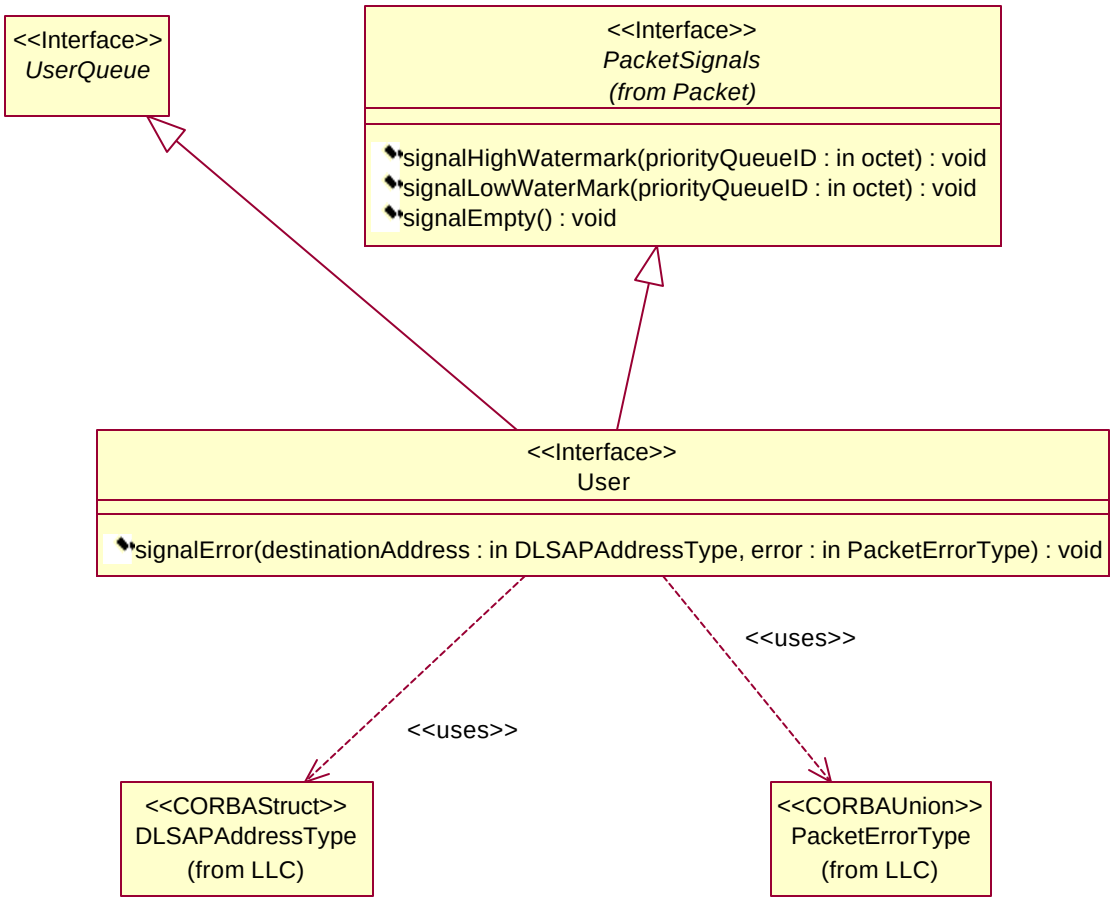

Figure 18. Connectionless Class Diagram: User Interface

G.4.2.1 DL_UNITDATA_REQUEST.

Conveys one DLSDU from the DLS user to the DLS provider for transmission to a peer DLS user.

Because connectionless data transfer is an unacknowledged service, the DLS provider makes no guarantees of delivery of connectionless SDUs. It is the responsibility of the DLS user to do any necessary sequencing or retransmission of SDUs in the event of a presumed loss.

G.4.2.1.1 Synopsis.

```
oneway void pushPacket (
        in octet                priority,
        in RequestHeaderType    control,
        in CF::OctetSequence    payload
);
```

G.4.2.1.2 Parameters.

priority

priority for queuing in the DLS provider. (0 is lowest priority)

control

control provides information other than user data and is of the following structure.

```
struct RequestHeaderType {
        DLSAPAddressType  destinationAddress;
        unsigned long     priority;
};
```

destinationAddress

conveys the DLSAP address of the destination DLS user. If the destination user is implemented using this service, this address is the full DLSAP address returned from the DL_BIND_REQ.

priority

indicates the priority value within the supported range for this particular SDU.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the attribute maxSDU in the DL_MAX_SDU primitive.

G.4.2.1.3 State.

The primitive is valid in STATE_IDLE.

G.4.2.1.4 New State.

The resulting state is unchanged.

G.4.2.1.5 Response.

If the DLS provider accepts the data for transmission, there is no response. This does not, however, guarantee that the data will be delivered to the destination Service User, since the connectionless data transfer is not a confirmed service. If the request is erroneous, message DL_UDERROR_IND is returned, and the resulting state is unchanged. If for some reason the request cannot be processed, the DLS provider may generate a DL_UDERROR_IND to report the problem. There is, however, no guarantee that such an error report will be generated for all undeliverable data units, since connectionless data transfer is not a confirmed service.

G.4.2.1.6 Originator.

This primitive is initiated by the Service User.

G.4.2.1.7 Errors/Exceptions.

The following errors can be returned in the DL_UDERROR_IND primitive:

ERROR_BADADDR The destination DLSAP address was in an incorrect format or

	contained invalid information
ERROR_BADDATA	The amount of data in the current SDU exceeded the DLS provider's SDU limit.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_UNSUPPORTED	Requested priority not supplied by provider.

G.4.2.2 DL_UNITDATA_IND.

Conveys one SDU from the DLS provider to the DLS user.

G.4.2.2.1 Synopsis.

```
oneway void pushPacket (
        in octet                priority
        in IndicatorHeaderType  control,
        in CF::OctetSequence    payload
);
```

G.4.2.2.2 Parameters.

priority

priority for queuing in the DLS user. (0 is lowest priority)

control

```
struct IndicatorHeaderType {
        DLSAPAddressType  destinationAddress;
        DLSAPAddressType  sourceAddress;
        Boolean           groupAddress;
};
```

destinationAddress

conveys the address of the DLSAP where this DL_UNITDATA_IND is intended to be delivered.

sourceAddress

conveys the DLSAP address of the sending DLS user.

isGroupAddress

is set by the DLS provider upon receiving and passing upstream a packet when the destination address of the packet is a multicast or broadcast address.

payload

Payload is a sequence of octets that represent the SDU. The length of the octet sequence must be limited to *maxPacketSize* - the size of the address in the destinationAddress field of Control.

G.4.2.2.3 State.

The valid states in which this primitive may execute can only be enumerated in a complete API of which this building block is a part.

G.4.2.2.4 New State.

The resulting state is unchanged.

G.4.2.2.5 Response.

N/A.

G.4.2.2.6 Originator.

This primitive is initiated by the DLS provider.

G.4.2.2.7 Errors/Exceptions.

N/A.

G.4.2.3 DL_UDERROR_IND.

G.4.2.3.1 Synopsis.

```
oneway void signalError (
        in DLSAPAddressType     destinationAddress,
        in PacketErrorType      error
);
```

G.4.2.3.2 Parameters.

destinationAddress

conveys the address of the destination Service User.

error

conveys the error that occurred. It is a union of the following composition.

```
union PacketErrorType switch (boolean) {
        case TRUE: ServiceErrorType     usageError;
        case FALSE: unsigned long       errNo;
};
```

usageError

conveys an error indication that is service oriented. Valid if switch is TRUE. See **Errors/Exceptions** in the description of DL_UNITDATA_REQ for the error codes that apply to an erroneous DL_UNITDATA_REQ. In addition, the error value DL_UNDELIVERABLE may be returned if the request was valid but for some reason the DLS provider could not deliver the data unit (e.g. due to lack of sufficient local buffering to store the data unit). There is, however, no guarantee that such an error report will be generated for all undeliverable data units, since connectionless data transfer is not a confirmed service.

errNo

conveys a POSIX errno. Valid if switch is false.

G.4.2.3.3 State.

The primitive is valid in STATE_IDLE.

G.4.2.3.4 New State.

The resulting state is unchanged.

G.4.2.3.5 Response.

N/A.

G.4.2.3.6 Originator.

This primitive is initiated by the DLS provider.

G.4.2.3.7 Errors/Exceptions.

N/A.

G.4.3 acknowledged connectionless-mode service primitives.

Figure 19, Figure 20 and Figure 21are the type, Packet Building Block instantiations and Provider and User class diagrams respectively for the connectionless data transfer service. The DLS user implements the Provider interface and the DLS user implements the User interface.

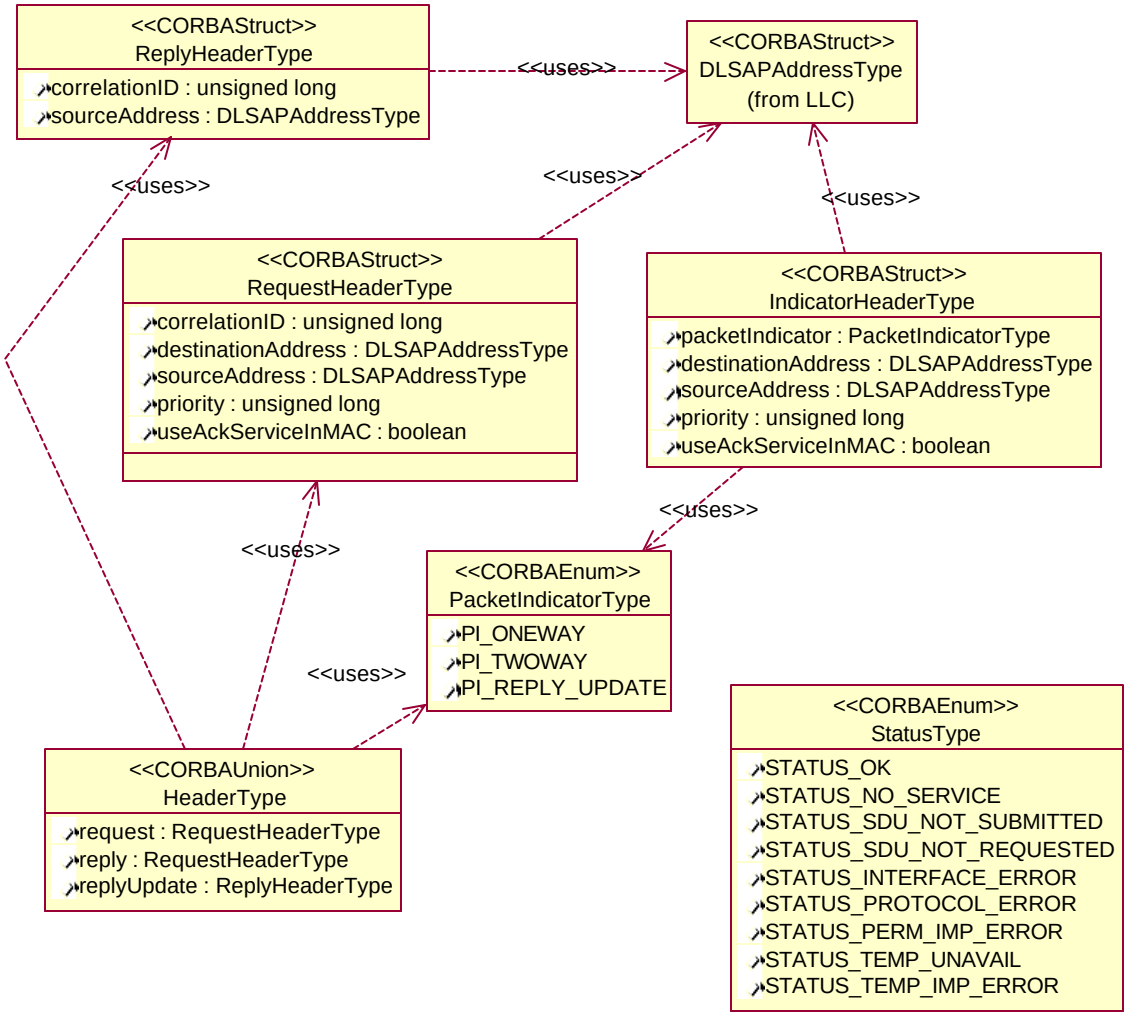

Figure 19. Acknowledged Connectionless Class Diagram: Types

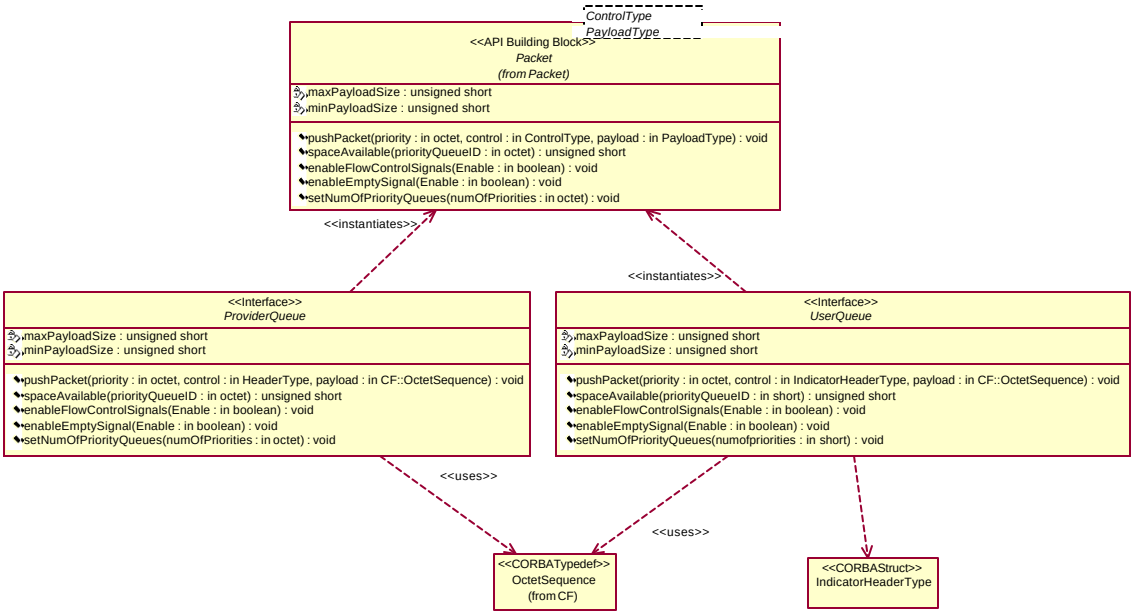

Figure 20. Acknowledged Connectionless Class Diagram: Packet Building Block Instantiations

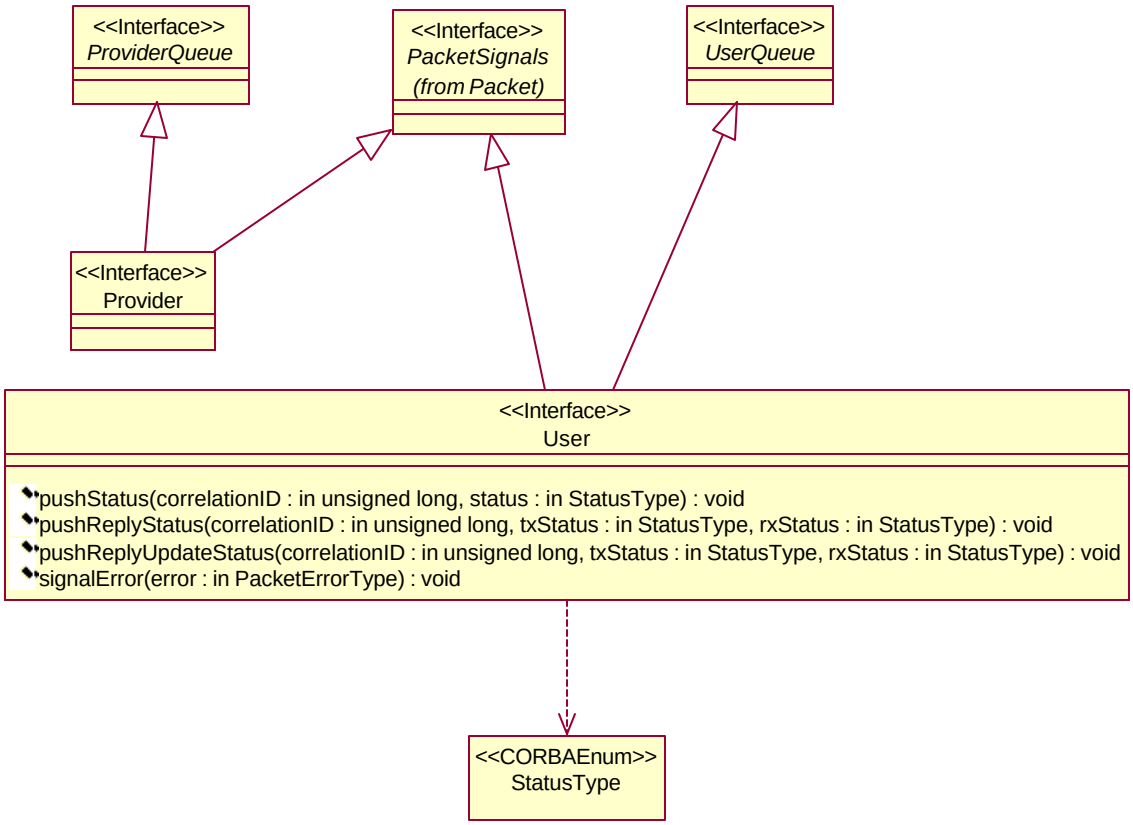

Figure 21. Acknowledged Connectionless Class Diagram: Provider and User Interfaces

G.4.3.1 DL_DATA_ACK_REQ.

This request is passed to the Data Link provider to request that a SDU be sent to a peer DLS user using acknowledged connectionless mode data unit transmission procedures.

G.4.3.1.1 Synopsis.

```
void pushPacket (
        in octet                priority,
        in HeaderType           control,
        in CF::OctetSequence    payload
);
```

G.4.3.1.2 Parameters.

priority

priority for queuing in the DLS provider. (0 is lowest priority)

control

control provides information other than user data and is of the following structure.

```
union HeaderType switch(PacketIndicatorType) {
case PI_ONEWAY: RequestHeaderType          request;
case PI_TWOWAY: RequestHeaderType          reply;
case PI_REPLY_UPDATE: ReplyHeaderType      replyUpdate;
};
```

packetIndicatorType

indicates the type of packet. Must be set to PI_ONEWAY for this primitive which indicates this is a packet that is unsolicited by the peer DLS provider and no reply is expected.

request

holds the header information for a packet to be sent for which no reply is expected and is of the following structure.

```
struct RequestHeaderType {
        unsigned long       correlationID;
        DLSAPAddressType    destinationAddress;
        DLSAPAddressType    sourceAddress;
        unsigned long       priority;
        boolean             useAckServiceInMAC;
};
```

correlationID

Conveys a unique identifier which will be returned in the DL_DATA_ACK_STATUS_IND primitive to allow the DLS User to correlate the status to the appropriate DL_DATA_ACK_REQ primitive.

destinationAddress

conveys the DLSAP address of the destination DLS user. If the destination user is implemented using this service, this address is the full DLSAP address returned from the DL_BIND_REQ.

sourceAddress

conveys the DLSAP address of the source DLS user.

priority

indicates the priority value within the supported range for this particular SDU.

useAckServiceInMAC

Specifies whether or not an acknowledge capability in the medium access control sublayer is to be used for the data unit transmission.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the DL_MAX_SDU primitive.

G.4.3.1.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.1.4 New State.

The resulting state is unchanged.

G.4.3.1.5 Response.

If the request is erroneous, message DL_ERROR_ACK is returned, and the resulting state is unchanged.

If the DLS Provider accepts the data for transmission, a DL_DATA_ACK_STATUS_IND is returned. This indication will indicate the success or failure of the data transmission. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station.

G.4.3.1.6 Originator.

This primitive is initiated by the DLS user.

G.4.3.1.7 Errors/Exceptions.

The following errors can be issued in a DL_ERROR_ACK primitive.

ERROR_BADADDR	The destination DLSAP address information was invalid or was in an incorrect format.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_NOT_SUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_UNSUPPORTED	Requested service or priority not supported by Provider (Request with response at the Medium Access Control sublayer).
ERROR_BAD_DATA	The amount of data in the current SDU exceeded the DLS provider’s SDU limit.

G.4.3.2 DL_DATA_ACK_IND

Conveys one DLSDU from the DLS Provider to the DLS User. This primitive indicates the arrival of a non-null, non-duplicate DLSDU from a peer Data Link User entity.

G.4.3.2.1 Synopsis.

```
void pushPacket (
        in octet                priority,
        in IndicatorHeaderType  control,
        in CF::OctetSequence    payload
);
```

G.4.3.2.2 Parameters.

priority

priority for queuing in the DLS user. (0 is lowest priority)

control

control provides information other than user data and is of the following structure.

```
struct IndicatorHeaderType {
        PacketIndicatorType   packetIndicator;
        DLSAPAddressType      destinationAddress;
        DLSAPAddressType      sourceAddress;
        unsigned long         priority;
        boolean               useAckServiceInMAC;
};
```

packetIndicator

indicates the type of packet. Must be set to PI_ONEWAY for this primitive which indicates this is a packet that has arrived unsolicited from a peer DLS provider.

destinationAddress

conveys the DLSAP address of the destination DLS user. If the destination user is implemented using this service, this address is the full DLSAP. It is the address returned in the DL_BIND_REQ primitive.

sourceAddress

conveys the DLSAP address of the source DLS user.

priority

priority provided for the data unit transmission.

useAckServiceInMAC

Specifies whether or not an acknowledge capability in the medium access control sublayer is to be used for the data unit transmission.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the DL_MAX_SDU primitive.

.

G.4.3.2.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.2.4 New State.

The resulting state is unchanged.

G.4.3.2.5 Response.

N/A.

G.4.3.2.6 Originator.

This primitive is initiated by the DLS provider.

G.4.3.2.7 Errors/Exceptions.

N/A.

G.4.3.3 DL_DATA_ACK_STATUS_IND

Conveys the results of the previous associated DL_DATA_ACK_REQ from the DLS Provider to the DLS user.

G.4.3.3.1 Synopsis.

```
oneway void pushStatus (
        in unsigned long    correlationID,
        in StatusType       status
);
```

G.4.3.3.2 Parameters.

correlationID

conveys the unique identifier passed with the DL_DATA_ACK_REQ primitive, to allow the DLS user correlate the status to the appropriate DL_DATA_ACK_REQ.

status

indicates the success or failure of the previous associated acknowledged connectionless-mode data unit transmission request.

STATUS_OK	Command accepted.
STATUS _ NO_SERVICE	Unimplemented or inactivated service.
STATUS _ INTERFACE_ERROR	LLC User Interface error
STATUS _PROTOCOL_ERROR	Protocol error
STATUS _ PERM_IMP_ERROR	Permanent implementation dependent error
STATUS _ TEMP_UNAVAIL	Resources temporarily unavailable.
STATUS _ TEMP_IMP_ERROR	Temporary implementation dependent error.

G.4.3.3.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.3.4 New State.

The resulting state is unchanged.

G.4.3.3.5 Response.

N/A.

G.4.3.3.6 Originator.

This primitive is initiated by the DLS provider.

G.4.3.3.7 Errors/Exceptions.

N/A.

G.4.3.4 DL_REPLY_REQ

This request primitive is passed to the DLS provider by the DLS user to request that a SDU be returned from a peer DLS provider or that SDUs be exchanged between stations using acknowledged connectionless mode data unit exchange procedures.

G.4.3.4.1 Synopsis.

```
void pushPacket (
        in octet                priority,
        in RequestHeaderType    control,
        in CF::OctetSequence    payload
);
```

G.4.3.4.2 Parameters.

priority

priority for queuing in the DLS provider. (0 is lowest priority)

control

control provides information other than user data and is of the following structure.

```
union HeaderType switch(PacketIndicatorType) {
case PI_ONEWAY: RequestHeaderType          request;
case PI_TWOWAY: RequestHeaderType          replyRequest;
case PI_REPLY_UPDATE: ReplyHeaderType      replyUpdate;
};
```

packetIndicatorType

indicates the type of packet. Must be set to PI_TWOWAY for this primitive which indicates this is a packet for which a reply is expected from the peer DLS provider.

replyRequest

holds the header information for a packet to be sent for which a reply is expected and is of the following structure.

```
struct RequestHeaderType {
        unsigned long       correlationID;
        DLSAPAddressType    destinationAddress;
        DLSAPAddressType    sourceAddress;
        unsigned long       priority;
        boolean             useAckServiceInMAC;
};
```

correlationID

Conveys a unique identifier which will be returned in the DL_ REPLY_STATUS_IND primitive to allow the DLS User to correlate the status to the appropriate DL_REPLY_REQ primitive.

destinationAddress

conveys the DLSAP address of the destination DLS user. If the destination user is implemented using this service, this address is the full DLSAP address returned from the DL_BIND_REQ.

sourceAddress

conveys the DLSAP address of the source DLS user.

priority

indicates the priority value within the supported range for this particular SDU.

useAckServiceInMAC

Specifies whether or not an acknowledge capability in the medium access control sublayer is to be used for the data unit transmission.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the DL_MAX_SDU primitive.

G.4.3.4.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.4.4 New State.

The resulting state is unchanged.

G.4.3.4.5 Response.

If the request is erroneous, message DL_ERROR_ACK is returned, and the resulting state is unchanged.

If the DLS Provider accepts the data for transmission, a DL_REPLY_STATUS_IND is returned. This indication will indicate the success or failure of the data transmission. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station.

G.4.3.4.6 Originator.

This primitive is initiated by the DLS user.

G.4.3.4.7 Errors/Exceptions.

The following errors can be issued in a DL_ERROR_ACK primitive

ERROR_BADADDR	The destination DLSAP address information was invalid or was in an incorrect format.
ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_NOT_SUPPORTED	The primitive is known, but not supported by the DLS provider.
ERROR_UNSUPPORTED	Requested service or priority not supported by Provider (Request with response at the Medium Access Control sublayer).
ERROR_BAD_DATA	The amount of data in the current SDU exceeded the DLS provider’s SDU limit.

G.4.3.5 DL_REPLY_IND

This primitive is the service indication primitive for the acknowledged connectionless-mode data unit exchange service. It is passed from the DLS provider to the DLS user to indicate either a successful request of a SDU from the peer data link user entity, or exchange of SDUs with a peer data link user entity.

G.4.3.5.1 Synopsis.

```
void pushPacket (
        in octet                    priority,
        in IndicatorHeaderType      control,
        in CF::OctetSequence        payload
);
```

G.4.3.5.2 Parameters.

priority

priority for queuing in the DLS user. (0 is lowest priority)

control

control provides information other than user data and is of the following structure.

```
struct IndicatorHeaderType {
        PacketIndicatorType   packetIndicator;
        DLSAPAddressType      destinationAddress;
        DLSAPAddressType      sourceAddress;
        unsigned long         priority;
        boolean               useAckServiceInMAC;
};
```

packetIndicator

indicates the type of packet. Must be set to PI_TWOWAY for this primitive.

destinationAddress

conveys the DLSAP address of the destination DLS user. If the destination user is implemented using this service, this address is the full DLSAP. It is the address returned in the DL_BIND_REQ primitive.

sourceAddress

conveys the DLSAP address of the source DLS user.

priority

priority provided for the data unit transmission.

useAckServiceInMAC

Specifies whether or not an acknowledge capability in the medium access control sublayer is to be used for the data unit transmission.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the DL_MAX_SDU primitive.

G.4.3.5.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.5.4 New State.

The resulting state is unchanged.

G.4.3.5.5 Response.

N/A.

G.4.3.5.6 Originator.

This primitive is initiated by the DLS provider.

G.4.3.5.7 Errors/Exceptions.

N/A.

G.4.3.6 DL_REPLY_STATUS_IND

Conveys the results of the previous associated DL_REPLY_REQ from the DLS provider to the DLS user.

G.4.3.6.1 Synopsis.

```
oneway void pushReplyStatus (
        in unsigned long    correlationID,
        in StatusType       status
);
```

G.4.3.6.2 Parameters.

correlationID

conveys the unique identifier passed with the DL_REPLY_REQ primitive, to allow the DLS user correlate the status to the appropriate DL_REPLY_REQ.

txStatus

indicates the success or failure of the transmission portion of the previous associated acknowledged connectionless-mode data unit transmission request.

STATUS_OK	Command accepted.
STATUS_NO_SERVICE	Unimplemented or inactivated service.
STATUS_INTERFACE_ERROR	LLC User Interface error
STATUS_PROTOCOL_ERROR	Protocol error
STATUS_PERM_IMP_ERROR	Permanent implementation dependent error
STATUS _TEMP_UNAVAIL	Resources temporarily unavailable.
STATUS_TEMP_IMP_ERROR	Temporary implementation dependent error.

rxStatus

indicates the success or failure of the reception portion of the previous associated acknowledged connectionless-mode data unit exchange request.

STATUS_OK	Response DLSDU present.
STATUS_NO_SERVICE	Unimplemented or inactivated service.
STATUS_SDU_NOT_SUBMITTED	Response DLSDU never submitted.
STATUS_SDU_NOT_REQUESTED	Response DLSDU not requested.
STATUS _ INTERFACE_ERROR	LLC User Interface error
STATUS _ PROTOCOL_ERROR	Protocol error

STATUS _ PERM_IMP_ERROR	Permanent implementation dependent error
STATUS _ TEMP_UNAVAIL	Resources temporarily unavailable.
STATUS _ TEMP_IMP_ERROR	Temporary implementation dependent error.

G.4.3.6.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.6.4 New State.

The resulting state is unchanged.

G.4.3.6.5 Response.

N/A.

G.4.3.6.6 Originator.

This primitive is initiated by the DLS provider.

G.4.3.6.7 Errors/Exceptions.

N/A.

G.4.3.7 DL_REPLY_UPDATE_REQ.

Conveys a SDU to the DLS provider from the DLS user to be held by the DLS provider and sent out at a later time when requested to do so by the peer DLS provider.

G.4.3.7.1 Synopsis.

```
void pushPacket (
        in ReplyHeaderType   control,
        in CF::OctetSequence payload
);
```

G.4.3.7.2 Parameters.

control

control provides information other than user data and is of the following structure.

```
union HeaderType switch(PacketIndicatorType) {
case PI_ONEWAY: RequestHeaderType       request;
case PI_TWOWAY: RequestHeaderType       reply;
case PI_REPLY_UPDATE: ReplyHeaderType   replyUpdate;
};
```

packetIndicatorType

indicates the type of packet. Must be set to PI_REPLY_UPDATE for this primitive which indicates this is a packet that is to be held by the DLS provider until requested to be sent by a peer DLS user.

replyUpdate

holds the header information for the reply update.

```
struct ReplyHeaderType {
        unsigned long       correlationID;
        DLSAPAddressType    sourceAddress;
};
```

correlationID

Conveys a unique identifier which will be returned in the DL_ REPLY_UPDATE_ IND primitive to allow the DLS User to correlate the status to the appropriate DL_REPLY_UPDATE_REQ primitive.

sourceAddress

conveys the DLSAP address of the source DLS user.

payload

is a sequence of octets that represent the SDU. The amount of user data that may be transferred in a single SDU is limited. This limit is conveyed by the DL_MAX_SDU primitive.

.

G.4.3.7.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.7.4 New State.

The resulting state is unchanged.

G.4.3.7.5 Response.

If the request is erroneous, message DL_ERROR_ACK is returned, and the resulting state is unchanged.

If the DLS Provider accepts the data for transmission, a DL_REPLY_ UPDATE_ STATUS_IND is returned. This indication will indicate the success or failure of the data transmission. Although the exchange service is connectionless, in-sequence delivery is guaranteed for data sent by the initiating station.

G.4.3.7.6 Originator.

This primitive is initiated by the DLS user.

G.4.3.7.7 Errors/Exceptions.

The following errors can be issued in a DL_ERROR_ACK primitive

ERROR_INVALID_STATE	The primitive was issued from an invalid state.
ERROR_UNSUPPORTED	Requested service or priority not supported by Provider (Request with response at the Medium Access Control sublayer).
ERROR_BAD_DATA	The amount of data in the current SDU exceeded the DLS provider's SDU limit.

G.4.3.8 DL_REPLY_UPDATE_STATUS_IND.

Conveys the results of the previous associated DL_REPLY_REQ from the DLS provider to the DLS user.

G.4.3.8.1 Synopsis.

```
oneway void pushReplyUpdateStatus (
        in unsigned long    correlationID,
        in StatusType       status
);
```

G.4.3.8.2 Parameters.

correlationID

conveys the unique identifier passed with the DL_REPLY_UPDATE_REQ primitive, to allow the DLS user correlate the status to the appropriate DL_REPLY_UPDATE_REQ.

txStatus

indicates the success or failure of the transmission portion of the previous associated acknowledged connectionless-mode data unit transmission request.

STATUS_OK	Command accepted.
STATUS_NO_SERVICE	Unimplemented or inactivated service.
STATUS_INTERFACE_ERROR	LLC User Interface error
STATUS_PROTOCOL_ERROR	Protocol error
STATUS_PERM_IMP_ERROR	Permanent implementation dependent error
STATUS _TEMP_UNAVAIL	Resources temporarily unavailable.
STATUS_TEMP_IMP_ERROR	Temporary implementation dependent error.

rxStatus

indicates the success or failure of the reception portion of the previous associated acknowledged connectionless-mode data unit exchange request.

STATUS_OK	Response DLSDU present.
STATUS_NO_SERVICE	Unimplemented or inactivated service.
STATUS_SDU_NOT_SUBMITTED	Response DLSDU never submitted.
STATUS_SDU_NOT_REQUESTED	Response DLSDU not requested.
STATUS _ INTERFACE_ERROR	LLC User Interface error
STATUS _ PROTOCOL_ERROR	Protocol error

STATUS _ PERM_IMP_ERROR	Permanent implementation dependent error
STATUS _ TEMP_UNAVAIL	Resources temporarily unavailable.
STATUS _ TEMP_IMP_ERROR	Temporary implementation dependent error.

G.4.3.8.3 State.

The primitive is valid in STATE_IDLE.

G.4.3.8.4 New State.

The resulting state is unchanged.

G.4.3.8.5 Response.

N/A.

G.4.3.8.6 Originator.

This primitive is initiated by the DLS provider.

G.4.3.8.7 Errors/Exceptions.

N/A.

G.5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

TBD.

G.6 PRECEDENCE OF SERVICE PRIMITIVES.

Precedence of primitives is left to service definitions of full APIs.

G.7 SERVICE USER GUIDELINES.

Service user guidelines are left to service definitions of full APIs.

G.8 SERVICE PROVIDER-SPECIFIC INFORMATION.

This section will be filled in for complete APIs that inherit this building block.

G.9 IDL.

```
//Source file: llc.idl

#ifndef __LLC_DEFINED
#define __LLC_DEFINED

/* CmIdentification
  %X% %Q% %Z% %W% */

#include "packet.idl"
#include "cf.idl"


module LLC {

      enum ServiceErrorType {
            ERROR_BAD_SAP,
            ERROR_BAD_ADDRESS,
            ERROR_NO_ACCESS,
            ERROR_INVALID_STATE,
            ERROR_BAD_CORRELATION,
            ERROR_BAD_DATA,
            ERROR_UNSUPPORTED,
            ERROR_NOT_ENABLED,
            ERROR_TOO_MANY,
            ERROR_BOUND,
            ERROR_NO_AUTO,
            ERROR_NO_XIDAUTO,
            ERROR_NO_TESTAUTO,
            ERROR_NO_ADDRESS,
            ERROR_BAD_QOS_PARAMETERS,
            ERROR_UNDELIVERABLE
      };

      union PacketErrorType switch(boolean) {
            case TRUE: ServiceErrorType usageError;
            case FALSE: unsigned long errNo;
      };

      enum BindType {
            BIND_PEER,
            BIND_HIERARCHICAL
      };

      struct DLSAPAddressType {
            unsigned long sap;
            CF::OctetSequence address;
      };

      module Connectionless {

            struct RequestHeaderType {
                  DLSAPAddressType destinationAddress;
                  unsigned long priority;
            };
```

```
            struct IndicatorHeaderType {
                  DLSAPAddressType destinationAddress;
                  DLSAPAddressType sourceAddress;
                  boolean isGroupAddress;
            };

            interface ProviderQueue {
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  readonly attribute unsigned short maxPayloadSize;
                  readonly attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39BE5AAC0384 */
                  oneway void pushPacket (
                        in octet priority,
                        in RequestHeaderType control,
                        in CF::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39BE5AAC0398 */
                  unsigned long spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39BE5AAC03A3 */
                  oneway void enableFlowControlSignals (
                        in boolean Enable
                        );

/* This operation allows the client to turn the Empty Signal ON and OFF.
                  @roseuid 39BE5AAC03B6 */
                  oneway void enableEmptySignal (
                        in boolean Enable
                        );

                  /*
                  @roseuid 39BE5AAC03C1 */
                  oneway void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            interface UserQueue {
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */
```

```
                  readonly attribute unsigned short maxPayloadSize;
                  readonly attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39BE714B0359 */
                  oneway void pushPacket (
                        in octet priority,
                        in IndicatorHeaderType control,
                        in CF::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39BE714B036E */
                  unsigned short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39BE714B0381 */
                  oneway void enableFlowControlSignals (
                        in boolean Enable
                        );

/* This operation allows the client to turn the Empty Signal ON and OFF.
                  @roseuid 39BE714B038B */
                  oneway void enableEmptySignal (
                        in boolean Enable
                        );

                  /*
                  @roseuid 39BE714B039F */
                  oneway void setNumOfPriorityQueues (
                        in octet numofpriorities
                        );

            };

            interface Provider : Packet::PacketSignals, ProviderQueue {
            };

            interface User : UserQueue, Packet::PacketSignals {
                  /*
                  @roseuid 39C616CD03A6 */
                  oneway void signalError (
                        in DLSAPAddressType destinationAddress,
                        in PacketErrorType error
                        );

            };

      };
```

```
module LocalManagement {
      interface Provider;


      enum ServiceModeType {
            SM_CODLS,
            SM_CLDLS,
            SM_ACLDLS
      };

      enum StateType {
            STATE_UNATTACHED,
            STATE_UNBOUND,
            STATE_IDLE,
            STATE_UDQOS_PENDING,
            STATE_OUTCON_PENDING,
            STATE_INCON_PENDING,
            STATE_CONN_RES_PENDING,
            STATE_DATAXFER,
            STATE_USER_RESET_PENDING,
            STATE_PROV_RESET_PENDING,
            STATE_RESET_RES_PENDING,
            STATE_DISCON_PENDING_OUTCON,
            STATE_DISCON_PENDING_USER_RESET,
            STATE_DISCON_PENDING_DATAXFER,
            STATE_DISCON_PENDING_PROV_RESET
      };

      struct InfoType {
            StateType currentState;
            sequence <ServiceModeType> mode;
            CF::OctetSequence broadcastAddress;
            DLSAPAddressType address;
      };

      interface User {
            attribute string connectionID;
      };

      struct BindResponseType {
            DLSAPAddressType sapAddress;
            unsigned long maxConnectionInd;
            boolean autoXID;
            boolean autoTest;
      };

      enum PromiscuousModeType {
            PM_PHYSICAL,
            PM_SAP,
            PM_MULTI
      };

      struct BindRequestType {
            unsigned long sap;
            unsigned long maxConnectionInd;
            ServiceModeType serviceMode;
```

```
        boolean isListenStream;
        boolean autoXID;
        boolean autoTest;
};

interface Provider {
        exception InvalidPort {
        };

        exception ServiceUsageError {
                ServiceErrorType qualifier;
        };

        exception SystemError {
                unsigned long errNo;
        };

        readonly attribute unsigned long maxTU;
        readonly attribute unsigned long minTU;

        /*
        @roseuid 39B7AC4E00BA */
        void getInfo (
                in string connectionID,
                out InfoType info
                )
                raises (InvalidPort,SystemError);

        /*
        @roseuid 39B8EE8C008E */
        void bind (
                in string connectionID,
                in BindRequestType bindReq,
                out BindResponseType bindResp
                )
                raises (InvalidPort,ServiceUsageError,SystemError);

        /*
        @roseuid 39B929B60354 */
        void unbind (
                in string connectionID
                )
                raises (InvalidPort,ServiceUsageError,SystemError);

        /*
        @roseuid 39B92A5301D3 */
        void subsBind (
                in string connectionID,
                inout DLSAPAddressType address
                )
                raises (InvalidPort,ServiceUsageError,SystemError);

        /*
        @roseuid 39B92B0E039F */
        void subsUnbind (
                in string connectionID,
```

```
                  in DLSAPAddressType address
                  )
                  raises (InvalidPort,ServiceUsageError,SystemError);

            /*
            @roseuid 39B92B6201F1 */
            void enableMulticast (
                  in string connectionID,
                  in CF::OctetSequence address
                  )
                  raises (InvalidPort,ServiceUsageError,SystemError);

            /*
            @roseuid 39B92C820213 */
            void disableMulticast (
                  in string connectionID,
                  in CF::OctetSequence address
                  )
                  raises (InvalidPort,ServiceUsageError,SystemError);

            /*
            @roseuid 39B92D5B019D */
            void enablePromiscuousMode (
                  in string connectionID,
                  in PromiscuousModeType level
                  )
                  raises (InvalidPort,ServiceUsageError,SystemError);

            /*
            @roseuid 39B92D7502E5 */
            void disablePromiscuousMode (
                  in string connectionID,
                  in PromiscuousModeType level
                  )
                  raises (InvalidPort,ServiceUsageError,SystemError);

      };

};

module AckConnectionless {

      struct RequestHeaderType {
            unsigned long correlationID;
            DLSAPAddressType destinationAddress;
            DLSAPAddressType sourceAddress;
            unsigned long priority;
            boolean useAckServiceInMAC;
      };

      enum PacketIndicatorType {
            PI_ONEWAY,
            PI_TWOWAY,
            PI_REPLY_UPDATE
      };
```

```
            struct IndicatorHeaderType {
                  PacketIndicatorType packetIndicator;
                  DLSAPAddressType destinationAddress;
                  DLSAPAddressType sourceAddress;
                  unsigned long priority;
                  boolean useAckServiceInMAC;
            };

            interface UserQueue {
                  /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */

                  attribute unsigned short maxPayloadSize;
                  attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39BFC4CF0171 */
                  void pushPacket (
                        in octet priority,
                        in IndicatorHeaderType control,
                        in CF::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39BFC4CF0185 */
                  unsigned short spaceAvailable (
                        in short priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39BFC4CF0199 */
                  void enableFlowControlSignals (
                        in boolean Enable
                        );

/* This operation allows the client to turn the Empty Signal ON and OFF.
                  @roseuid 39BFC4CF01A3 */
                  void enableEmptySignal (
                        in boolean Enable
                        );

                  /*
                  @roseuid 39BFC4CF01AE */
                  void setNumOfPriorityQueues (
                        in short numofpriorities
                        );

            };

            enum StatusType {
                  STATUS_OK,
                  STATUS_NO_SERVICE,
```

```
            STATUS_SDU_NOT_SUBMITTED,
            STATUS_SDU_NOT_REQUESTED,
            STATUS_INTERFACE_ERROR,
            STATUS_PROTOCOL_ERROR,
            STATUS_PERM_IMP_ERROR,
            STATUS_TEMP_UNAVAIL,
            STATUS_TEMP_IMP_ERROR
      };

      interface User : Packet::PacketSignals, UserQueue {
            /*
            @roseuid 39BFF5DF0013 */
            oneway void pushStatus (
                  in unsigned long correlationID,
                  in StatusType status
                  );

            /*
            @roseuid 39CD02130382 */
            oneway void pushReplyStatus (
                  in unsigned long correlationID,
                  in StatusType txStatus,
                  in StatusType rxStatus
                  );

            /*
            @roseuid 39CD0933005A */
            oneway void pushReplyUpdateStatus (
                  in unsigned long correlationID,
                  in StatusType txStatus,
                  in StatusType rxStatus
                  );

            /*
            @roseuid 39C645A30310 */
            oneway void signalError (
                  in PacketErrorType error
                  );

      };

      struct ReplyHeaderType {
            unsigned long correlationID;
            DLSAPAddressType sourceAddress;
      };

      union HeaderType switch(PacketIndicatorType) {
            case PI_ONEWAY: RequestHeaderType request;
            case PI_TWOWAY: RequestHeaderType reply;
            case PI_REPLY_UPDATE: ReplyHeaderType replyUpdate;
      };

      interface ProviderQueue {
            /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the maximum number of
traffic units allowed in one pushPacket call.  */
```

```
                  readonly attribute unsigned short maxPayloadSize;
                  readonly attribute unsigned short minPayloadSize;

                  /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
                  @roseuid 39BFC2F800C9 */
                  void pushPacket (
                        in octet priority,
                        in HeaderType control,
                        in CF::OctetSequence payload
                        );

                  /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic Units.
                  @roseuid 39BFC2F800F2 */
                  unsigned short spaceAvailable (
                        in octet priorityQueueID
                        );

                  /* This operation allows the client to turn the High
Watermark Signal ON and OFF.
                  @roseuid 39BFC2F80105 */
                  void enableFlowControlSignals (
                        in boolean Enable
                        );

/* This operation allows the client to turn the Empty Signal ON and OFF.
                  @roseuid 39BFC2F80110 */
                  void enableEmptySignal (
                        in boolean Enable
                        );

                  /*
                  @roseuid 39BFC2F80123 */
                  void setNumOfPriorityQueues (
                        in octet numOfPriorities
                        );

            };

            interface Provider : ProviderQueue, Packet::PacketSignals {
            };

      };

};

#endif
```

G.10 UML.